

Kompilacja jądra Slackware

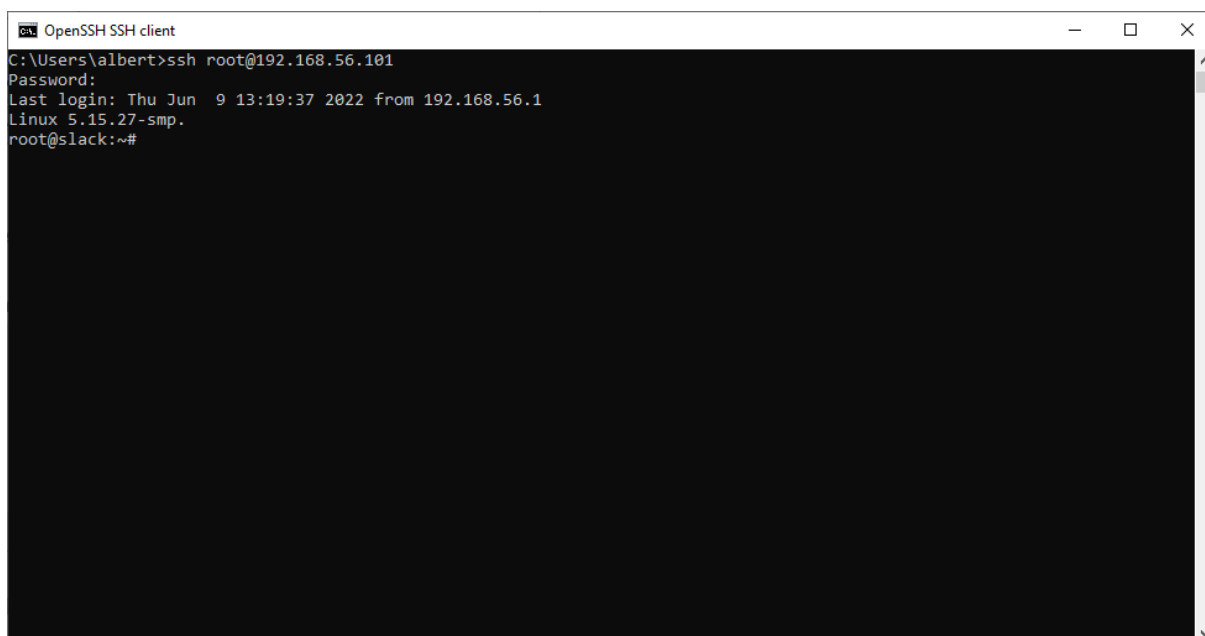
Albert Błaziak

16 czerwca 2022

Rozdział 1

Przygotowanie do kompilacji

Żeby ułatwić sobie pracę na początku połączyłem się do mojej maszyny wirtualnej za pomocą ssh, co przedstawia Rys. 1.1.



Rysunek 1.1: Połączenie za pomocą ssh

W trakcie przygotowania do kompilacji jądra na wirtualnej maszynie był zainstalowany Slackware 15.0 z wersją jądra 5.15.27 co prezentuje Rys. 1.2. Informacje te uzyskałem po wykonaniu poleceń `cat /proc/version` oraz `cat /etc/os-release`.

```
OpenSSH SSH client
root@slack:~# cat /proc/version
Linux version 5.15.27-smp (root@z-mp32.slackware.lan) (gcc (GCC) 11.2.0, GNU ld version 2.37-slack15) #1 SMP PREEMPT Tue
  Mar 8 20:11:16 CST 2022
root@slack:~# cat /etc/os-release
NAME=Slackware
VERSION="15.0"
ID=slackware
VERSION_ID=15.0
PRETTY_NAME="Slackware 15.0 i586"
ANSI_COLOR="0;34"
CPE_NAME="cpe:/o:slackware:slackware_linux:15.0"
HOME_URL="http://slackware.com/"
SUPPORT_URL="http://www.linuxquestions.org/questions/slackware-14/"
BUG_REPORT_URL="http://www.linuxquestions.org/questions/slackware-14/"
VERSION_CODENAME=stable
root@slack:~#
```

Rysunek 1.2: Aktualna wersja jądra

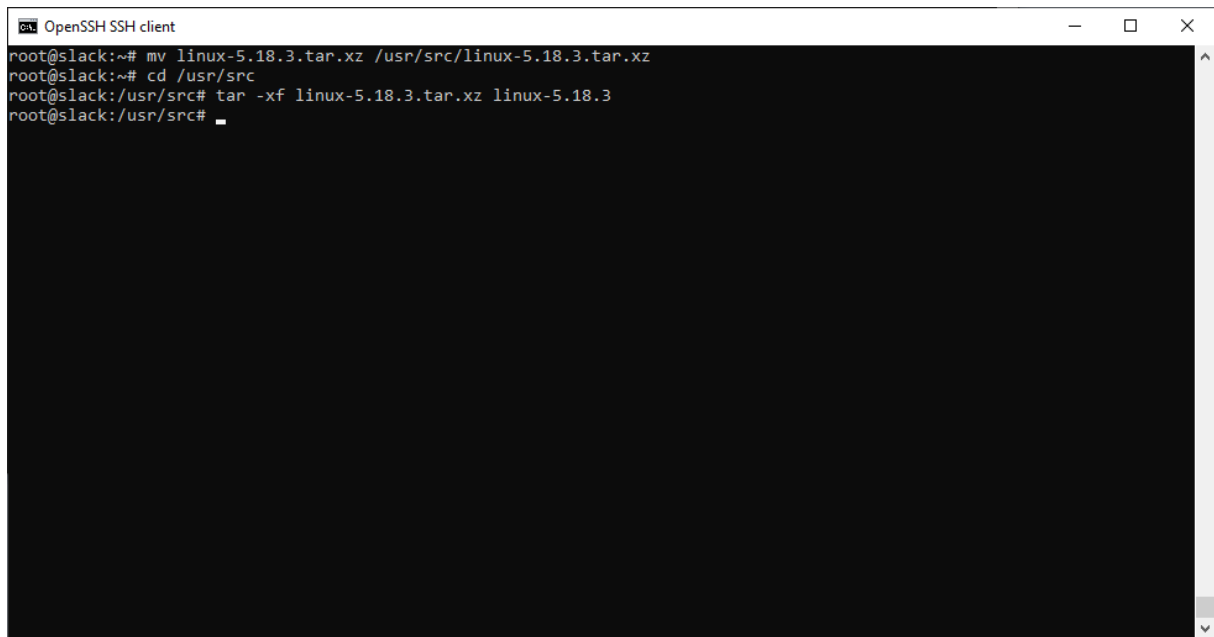
Najnowszą stabilną wersją dostępną na stronie *kernel.org* w momencie przygotowywania kompilacji była wersja **5.18.3**, którą pobrałem za pomocą narzędzia *wget* co prezentuje Rys. 1.3.

```
OpenSSH SSH client
root@slack:~# wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
--2022-06-09 12:41:51-- https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
Translacja cdn.kernel.org (cdn.kernel.org)... 151.101.113.176, 2a04:4e42:1b::432
Łączenie się z cdn.kernel.org (cdn.kernel.org)[151.101.113.176]:443... połączono.
Żądanie HTTP wysłano, oczekiwanie na odpowiedź... 200 OK
Długość: 129859840 (124M) [application/x-xz]
Zapis do: `linux-5.18.3.tar.xz'

linux-5.18.3.tar.xz          100%[=====>] 123,84M  18,9MB/s   w 6,2s
2022-06-09 12:41:57 (20,0 MB/s) - zapisano `linux-5.18.3.tar.xz' [129859840/129859840]
root@slack:~#
```

Rysunek 1.3: Wynik wykonania *wget*

Pliki jądra będą przechowywane w */usr/src*, dlatego też przeniosłem tam archiwum i tam je rozpakowałem co pokazane jest na Rys. 1.4



```
OpenSSH SSH client
root@slack:~# mv linux-5.18.3.tar.xz /usr/src/linux-5.18.3.tar.xz
root@slack:~# cd /usr/src
root@slack:/usr/src# tar -xf linux-5.18.3.tar.xz linux-5.18.3
root@slack:/usr/src#
```

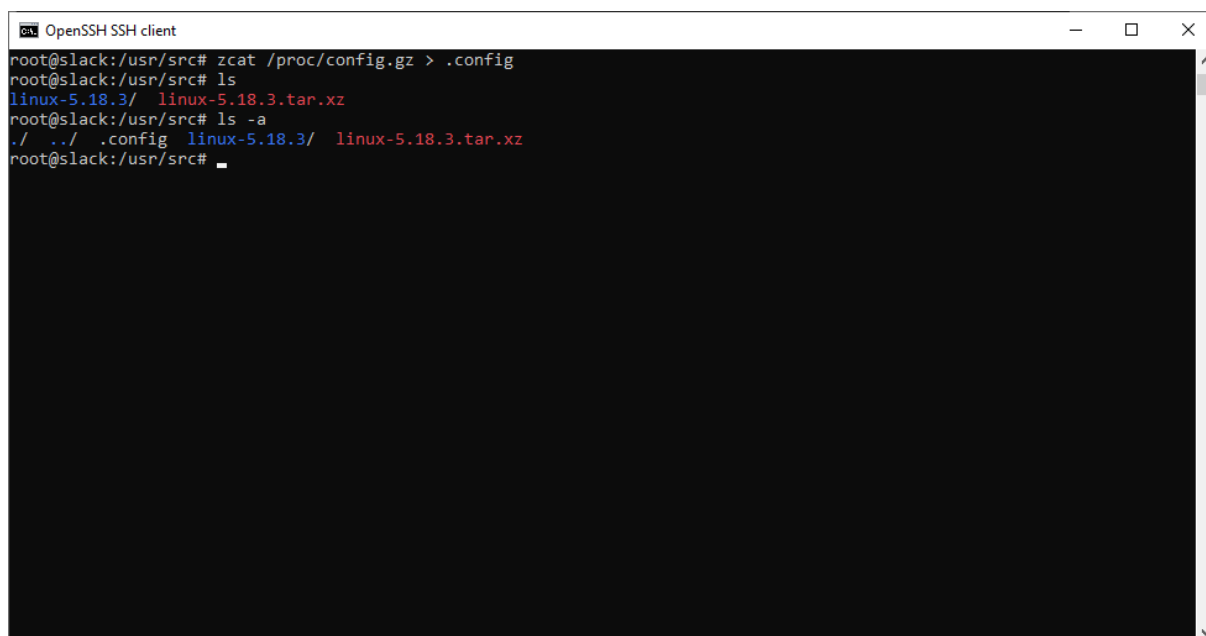
Rysunek 1.4: Przeniesienie i wypakowanie archiwum

Następnie możemy przejść do kompilacji jądra pierwszą metodą pokazaną w rozdziale 2.

Rozdział 2

Kompilacja jądra - old method

W pierwszej kolejności powinniśmy skopiować aktualny plik konfiguracyjny, co prezentuje wywołanie na Rys. 2.1



```
OpenSSH SSH client
root@slack:/usr/src# zcat /proc/config.gz > .config
root@slack:/usr/src# ls
linux-5.18.3/  linux-5.18.3.tar.xz
root@slack:/usr/src# ls -la
./  ../  .config  linux-5.18.3/  linux-5.18.3.tar.xz
root@slack:/usr/src#
```

Rysunek 2.1: Kopiowanie aktualnego pliku konfiguracyjnego

Następnie za pomocą polecenia *make localmodconfig* (Rys. 2.2) tworzymy nową konfigurację. Po pojawieniu się komunikatów dotyczących ustawień wszystkie zostawiłem na domyślne.

```
OpenSSH SSH client
# end of Kernel hacking
root@slack:/usr/src# cd linux-5.18.3
root@slack:/usr/src/linux-5.18.3# make localmodconfig
HOSTCC scripts/basic/fixdep
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: '/proc/config.gz'
*
* 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)
```

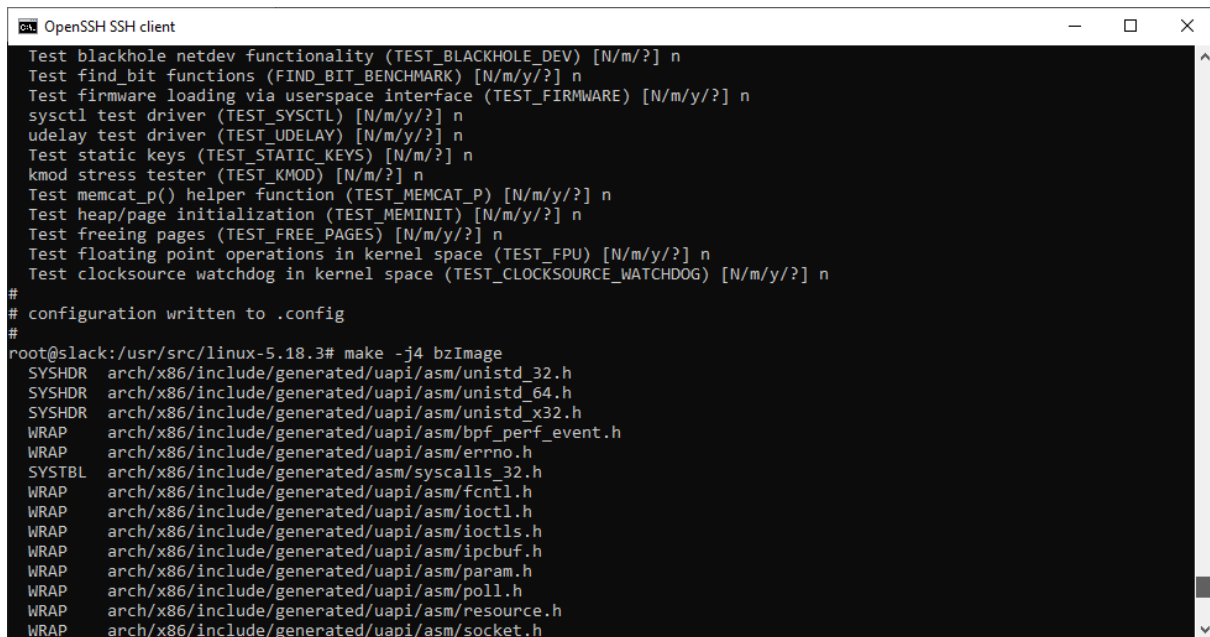
Rysunek 2.2: Wynik polecenia *make localmodconfig*

Po ustawieniu wszystkich opcji na domyślne otrzymałem komunikat o zapisaniu pliku do *.config*, Rys. 2.3.

```
OpenSSH SSH client
Test kstrtoint() 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 uuid 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
Test module loading with 'hello world' module (TEST_LKM) [N/m/?] n
Test module for compilation of bitops operations (TEST_BITOPS) [N/m/?] n
Test module for stress/performance analysis of vmalloc allocator (TEST_VMALLOCC) [N/m/?] n
Test user/kernel boundary protections (TEST_USER_COPY) [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
Test memcat_p() helper function (TEST_MEMCAT_P) [N/m/y/?] 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-5.18.3#
```

Rysunek 2.3: Konfiguracja została zapisana

Kiedy już mamy konfigurację można przejść do procesu kompilacji jądra. W tym celu skorzystałem z polecenia *make -j4 bzImage*. Skorzystałem z opcji *-j4*, ponieważ moja maszyna wirtualna ma przydzielone jedynie 2 rdzenie, więc to powinno przyspieszyć cały proces. Rozpoczęcie kompilacji prezentuje Rys. 2.4.

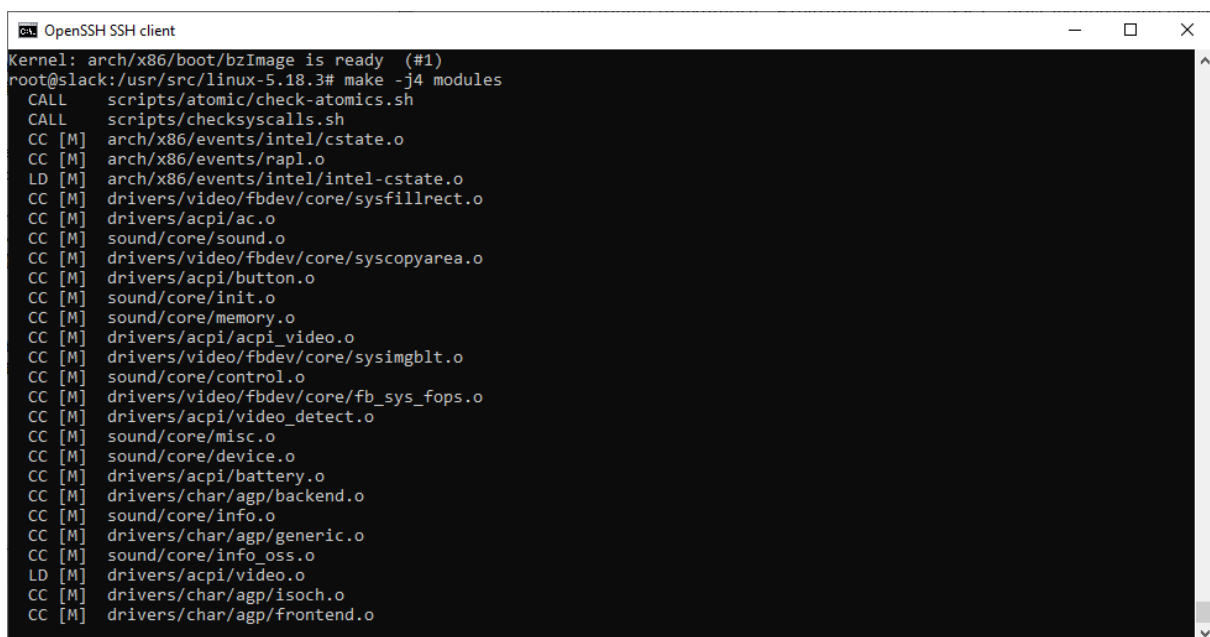


```
OpenSSH SSH client
Test blackhole netdev functionality (TEST_BLACKHOLE_DEV) [N/m/?] n
Test find_bit functions (FIND_BIT_BENCHMARK) [N/m/y/?] n
Test firmware loading via userspace interface (TEST_FIRMWARE) [N/m/y/?] n
sysctl test driver (TEST_SYSCTL) [N/m/y/?] n
udelay test driver (TEST_UDELAY) [N/m/y/?] n
Test static keys (TEST_STATIC_KEYS) [N/m/?] n
kmod stress tester (TEST_KMOD) [N/m/?] n
Test memcat_p() helper function (TEST_MEMCAT_P) [N/m/y/?] 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-5.18.3# make -j4 bzImage
SYSHDR arch/x86/include/generated/uapi/asm/unistd_32.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_64.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_x32.h
WRAP arch/x86/include/generated/uapi/asm/bpf_perf_event.h
WRAP arch/x86/include/generated/uapi/asm/errno.h
SYSTBL arch/x86/include/generated/asm/syscalls_32.h
WRAP arch/x86/include/generated/uapi/asm/fcntl.h
WRAP arch/x86/include/generated/uapi/asm/ioctl.h
WRAP arch/x86/include/generated/uapi/asm/ioctls.h
WRAP arch/x86/include/generated/uapi/asm/ipcbuf.h
WRAP arch/x86/include/generated/uapi/asm/param.h
WRAP arch/x86/include/generated/uapi/asm/poll.h
WRAP arch/x86/include/generated/uapi/asm/resource.h
WRAP arch/x86/include/generated/uapi/asm/socket.h
```

Rysunek 2.4: Rozpoczęcie procesu kompilacji

W trakcie procesu kompilacji jądra "uruchomiłem proces" oczekiwania, tworząc aplikację na zaliczenie przedmiotu "Programowanie w .NET" oraz jednocześnie śledząc na drugim monitorze przebieg kompilacji...

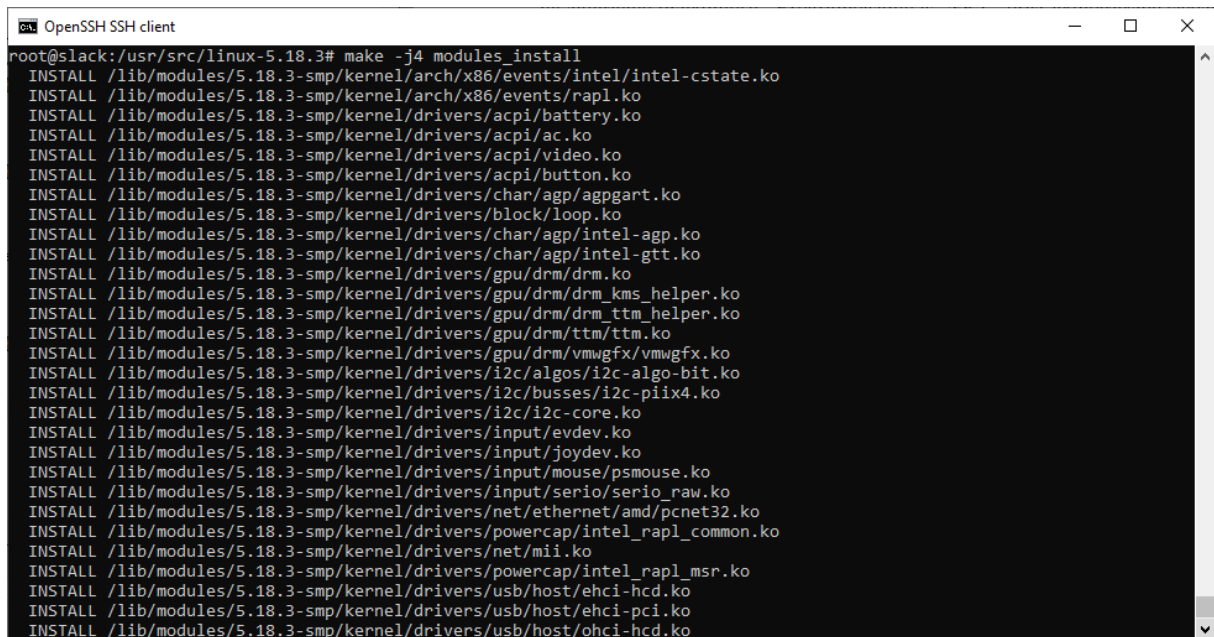
Ostatecznie okazało się, że proces kompilacji jądra nie zajął dużo czasu. Było to około 10-15 minut. Następnie skorzystałem z polecenia *make -j4 modules* w celu zbudowania modułów jądra. Ponownie skorzystałem z parametru *-j4*. Rozpoczęcie procesu budowania prezentuje Rys. 2.5.



```
OpenSSH SSH client
Kernel: arch/x86/boot/bzImage is ready (#1)
root@slack:/usr/src/linux-5.18.3# make -j4 modules
CALL scripts/atomic/check-atomics.sh
CALL scripts/checksyscalls.sh
CC [M] arch/x86/events/intel/cstate.o
CC [M] arch/x86/events/rapl.o
LD [M] arch/x86/events/intel/intel-cstate.o
CC [M] drivers/video/fbdev/core/sysfillrect.o
CC [M] drivers/acpi/ac.o
CC [M] sound/core/sound.o
CC [M] drivers/video/fbdev/core/syscopyarea.o
CC [M] drivers/acpi/button.o
CC [M] sound/core/init.o
CC [M] sound/core/memory.o
CC [M] drivers/acpi/acpi_video.o
CC [M] drivers/video/fbdev/core/sysimgblt.o
CC [M] sound/core/control.o
CC [M] drivers/video/fbdev/core/fb_sys_fops.o
CC [M] drivers/acpi/video_detect.o
CC [M] sound/core/misc.o
CC [M] sound/core/device.o
CC [M] drivers/acpi/battery.o
CC [M] drivers/char/agp/backend.o
CC [M] sound/core/info.o
CC [M] drivers/char/agp/generic.o
CC [M] sound/core/info_oss.o
LD [M] drivers/acpi/video.o
CC [M] drivers/char/agp/isoch.o
CC [M] drivers/char/agp/frontend.o
```

Rysunek 2.5: Rozpoczęcie procesu budowania modułów

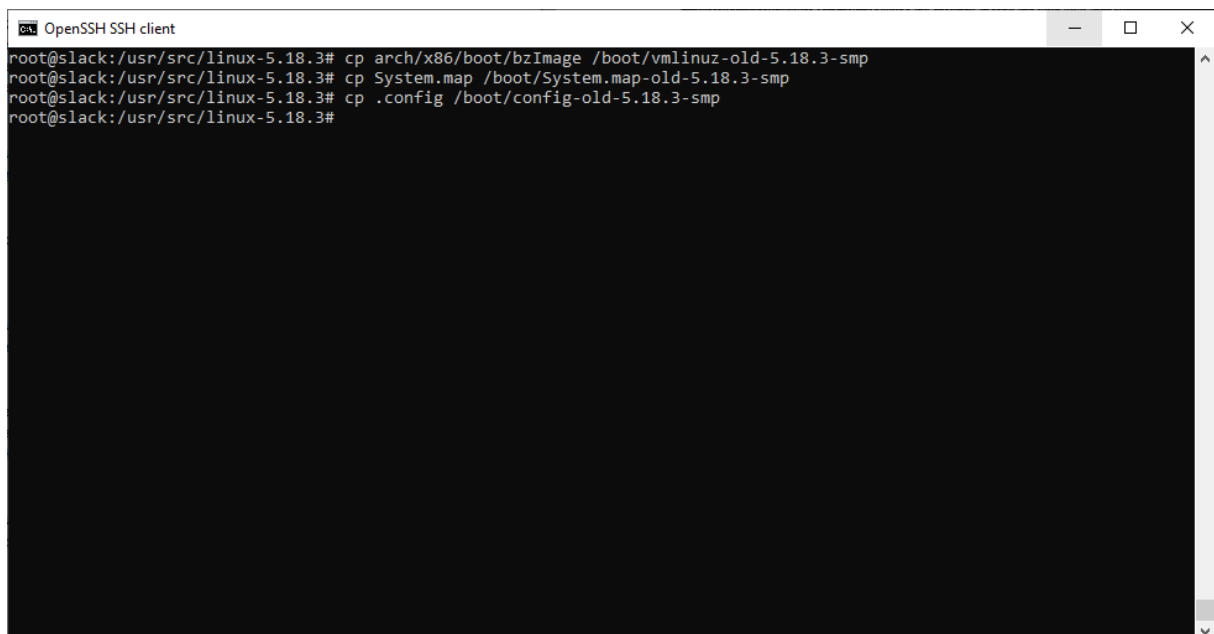
W celu zainstalowania zbudowanych modułów skorzystałem z `make -j4 modules_install` (Rys. 2.6).



```
root@slack:/usr/src/linux-5.18.3# make -j4 modules_install
INSTALL /lib/modules/5.18.3-smp/kernel/arch/x86/events/intel/intel-cstate.ko
INSTALL /lib/modules/5.18.3-smp/kernel/arch/x86/events/rapl.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/battery.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/ac.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/video.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/button.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/char/agp/agpgart.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/block/loop.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/char/agp/intel-agp.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/char/agp/intel-gtt.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/gpu/drm/drm.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/gpu/drm/drm_kms_helper.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/gpu/drm/drm_ttm_helper.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/gpu/drm/ttm/ttm.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/gpu/drm/vmwgfx/vmwgfx.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/i2c/algos/i2c-algo-bit.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/i2c/busses/i2c-piix4.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/i2c/i2c-core.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/input/evdev.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/input/joydev.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/input/mouse/psmouse.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/input/serio/serio_raw.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/net/ethernet/amd/pcnet32.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/powercap/intel_rapl_common.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/net/mii.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/powercap/intel_rapl_msr.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/usb/host/ehci-hcd.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/usb/host/ehci-pci.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/usb/host/ohci-hcd.ko
```

Rysunek 2.6: Rozpoczęcie procesu instalowania modułów

Po zainstalowaniu modułów należy przekopiować niezbędne pliki do katalogu `/boot`, aby mieć możliwość uruchomienia nowego jądra. Dodałem do nazwy plików prefix `old`, w celu identyfikacji plików jako utworzonych starą metodą. Proces prezentuje Rys. 2.7.

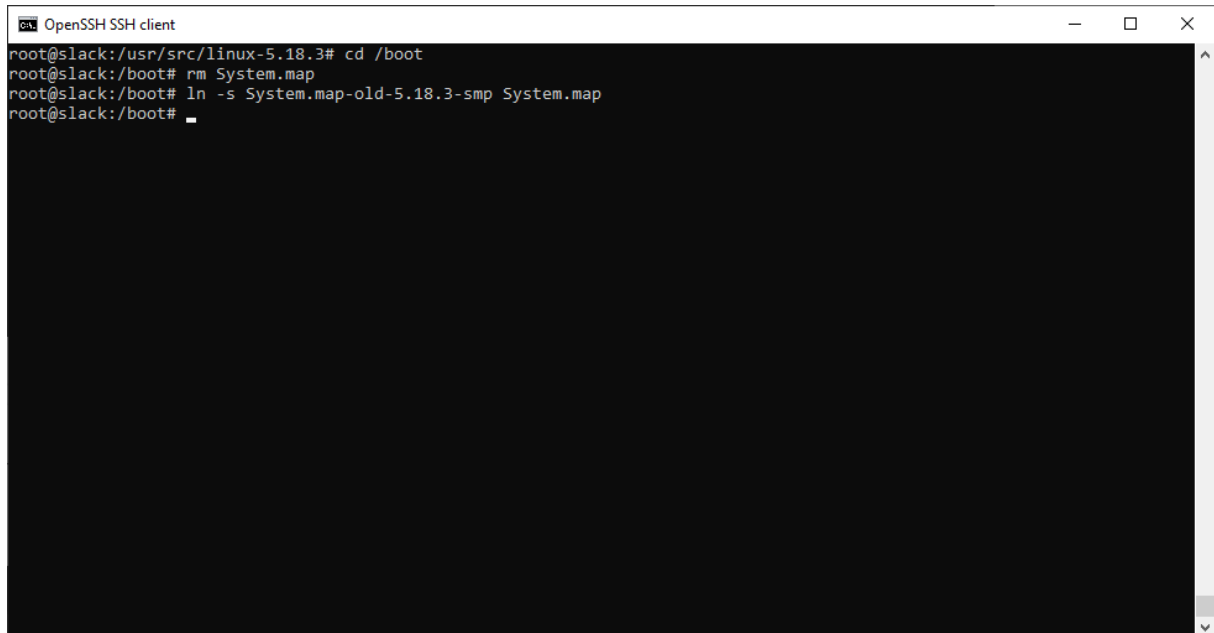


```
root@slack:/usr/src/linux-5.18.3# cp arch/x86/boot/bzImage /boot/vmlinuz-old-5.18.3-smp
root@slack:/usr/src/linux-5.18.3# cp System.map /boot/System.map-old-5.18.3-smp
root@slack:/usr/src/linux-5.18.3# cp .config /boot/config-old-5.18.3-smp
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 2.7: Kopia odpowiednich plików

Następnie należy zastąpić tablicę symboli `System.map` na nowo utworzoną. W tym celu usuwamy dotychczasowy plik i tworzymy dowiązanie symboliczne do pliku który skopio-

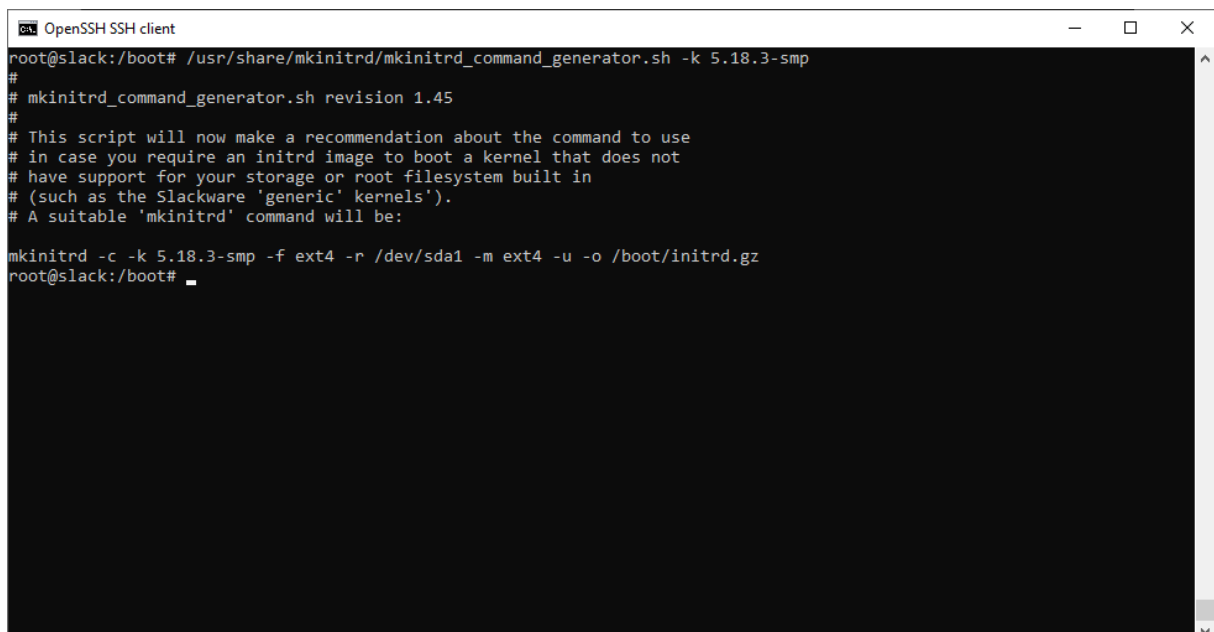
waliśmy (Rys. 2.8).



```
OpenSSH SSH client
root@slack:/usr/src/linux-5.18.3# cd /boot
root@slack:/boot# rm System.map
root@slack:/boot# ln -s System.map-old-5.18.3-smp System.map
root@slack:/boot#
```

Rysunek 2.8: Utworzenie dowiązania symbolicznego

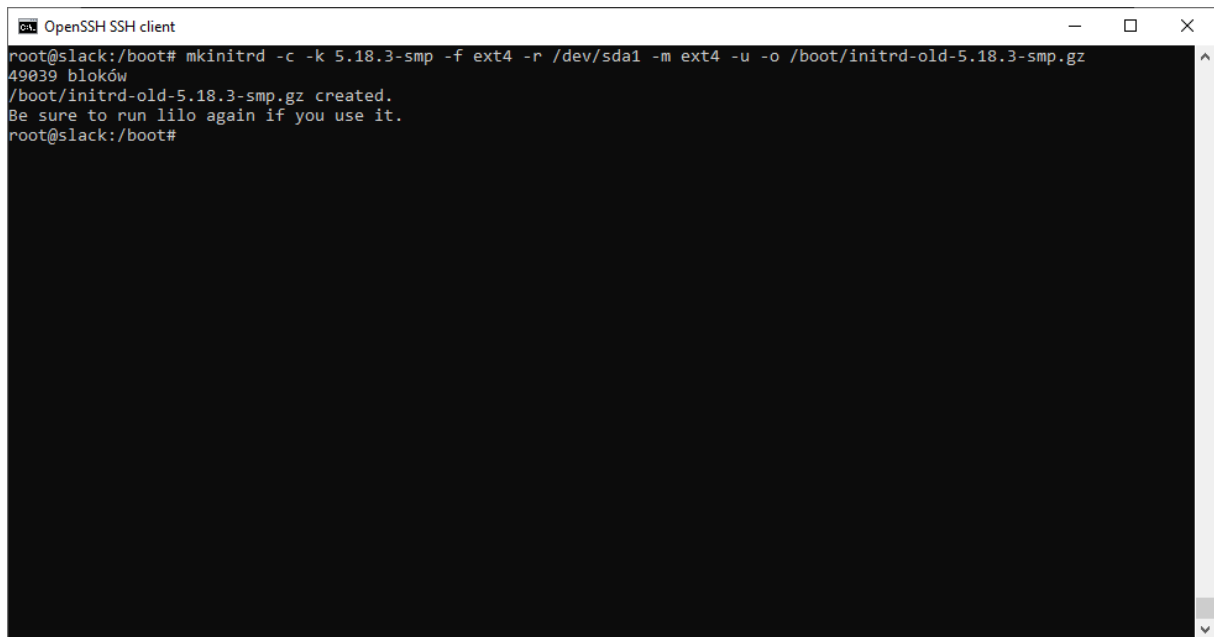
W kolejnym kroku za pomocą przeznaczonego do tego narzędzia generuję komendę, która pozwoli na utworzenie dysku RAM. Za pomocą opcji *-k* precyzuję wersję jądra (Rys 2.9).



```
OpenSSH SSH client
root@slack:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.18.3-smp
#
# mkinitrd_command_generator.sh revision 1.45
#
# This script will now make a recommendation about the command to use
# in case you require an initrd image to boot a kernel that does not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:
mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot#
```

Rysunek 2.9: Generowanie komendy do utworzenia dysku RAM

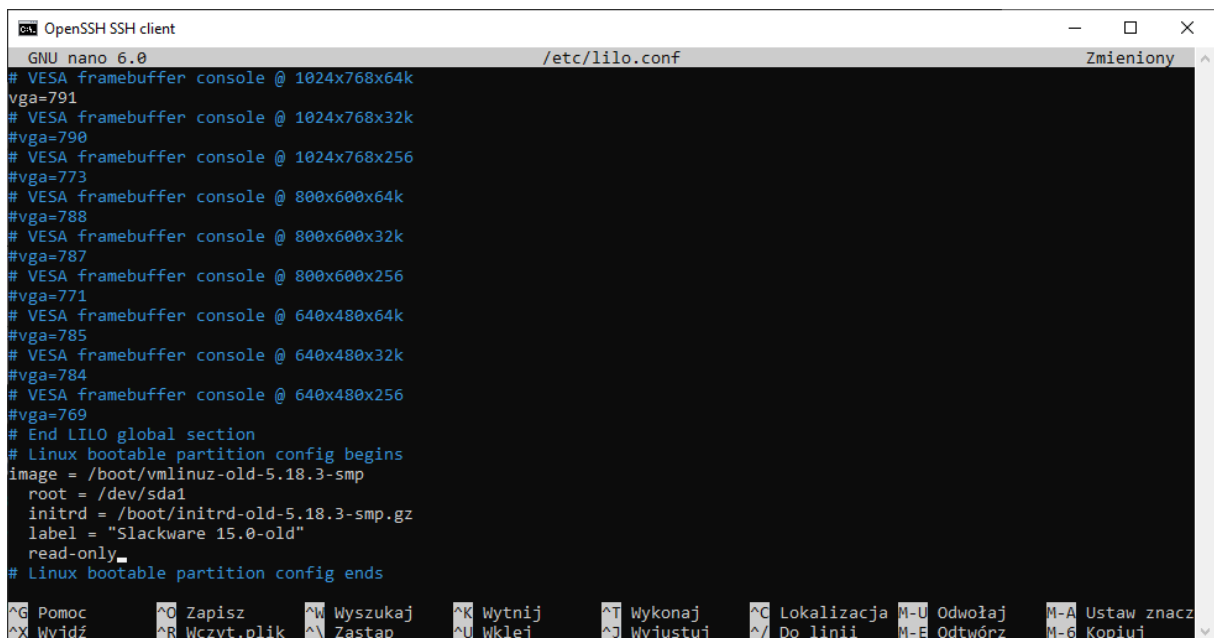
Przed wywołaniem wygenerowanego polecenia należy pamiętać o zmianie nazwy generowanego pliku, tak aby odpowiadał naszej wersji jądra. Wynik wygenerowanej komendy prezentuje Rys. 2.10.



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

Rysunek 2.10: Wywołanie komendy utworzenia dysku RAM

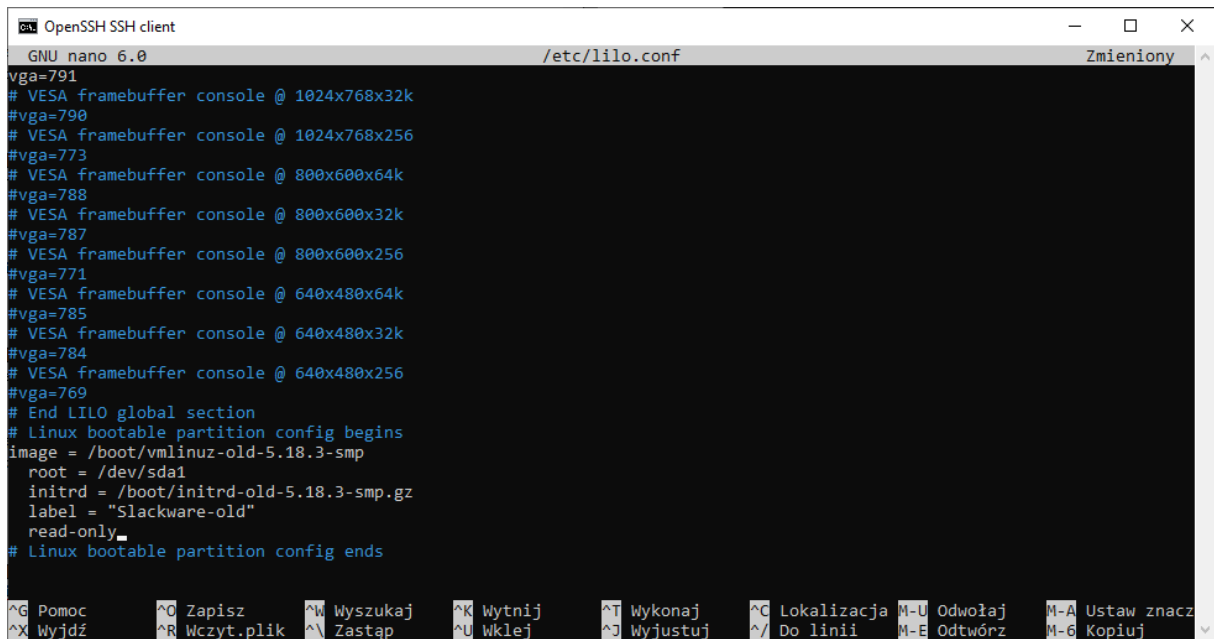
Dostajemy komunikat przypominający o konfiguracji lilo, więc to będzie następny krok. W tym celu za pomocą narzędzia nano edytuję plik `/etc/lilo.conf`. Ustawiam odpowiednią ścieżkę do obrazu oraz do dysku RAM oraz zmieniam etykietę. Prezentuje to Rys. 2.11.



```
GNU nano 6.0 /etc/lilo.conf Zmieniony
# VESA framebuffer console @ 1024x768x64k
vga=791
# 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-old-5.18.3-smp
root = /dev/sda1
initrd = /boot/initrd-old-5.18.3-smp.gz
label = "Slackware 15.0-old"
read-only
# Linux bootable partition config ends
```

Rysunek 2.11: Edycja lilo.conf

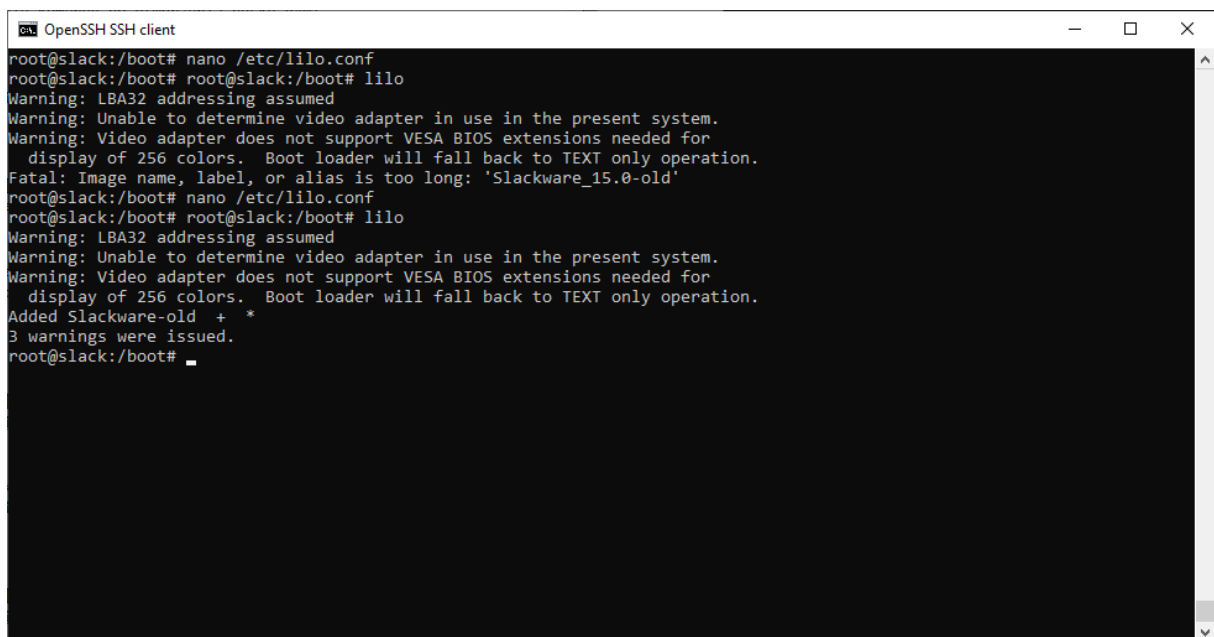
Po zapisaniu zmian uruchomiłem komendę `lilo` i okazało się, że etykieta, którą nadałem jest zbyt długa. Poprawiłem ją (Rys. 2.12) i uruchomiłem `lilo` jeszcze raz (Rys 2.13).



```
OpenSSH SSH client
GNU nano 6.0 /etc/lilo.conf Zmieniony
vga=791
# 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-old-5.18.3-smp
  root = /dev/sda1
  initrd = /boot/initrd-old-5.18.3-smp.gz
  label = "Slackware-old"
  read-only_
# Linux bootable partition config ends

^G Pomoc      ^O Zapisz      ^W Wyszukaj    ^K Wytnij     ^T Wykonaj    ^C Lokalizacja M-U Odwołaj      M-A Ustaw znacz
^X Wyjdź      ^R Wczyt.plik ^_ Zastąp      ^U Wklej      ^J Wyjustuj   ^_ Do linii    M-E Odtwórz     M-G Kopiuuj
```

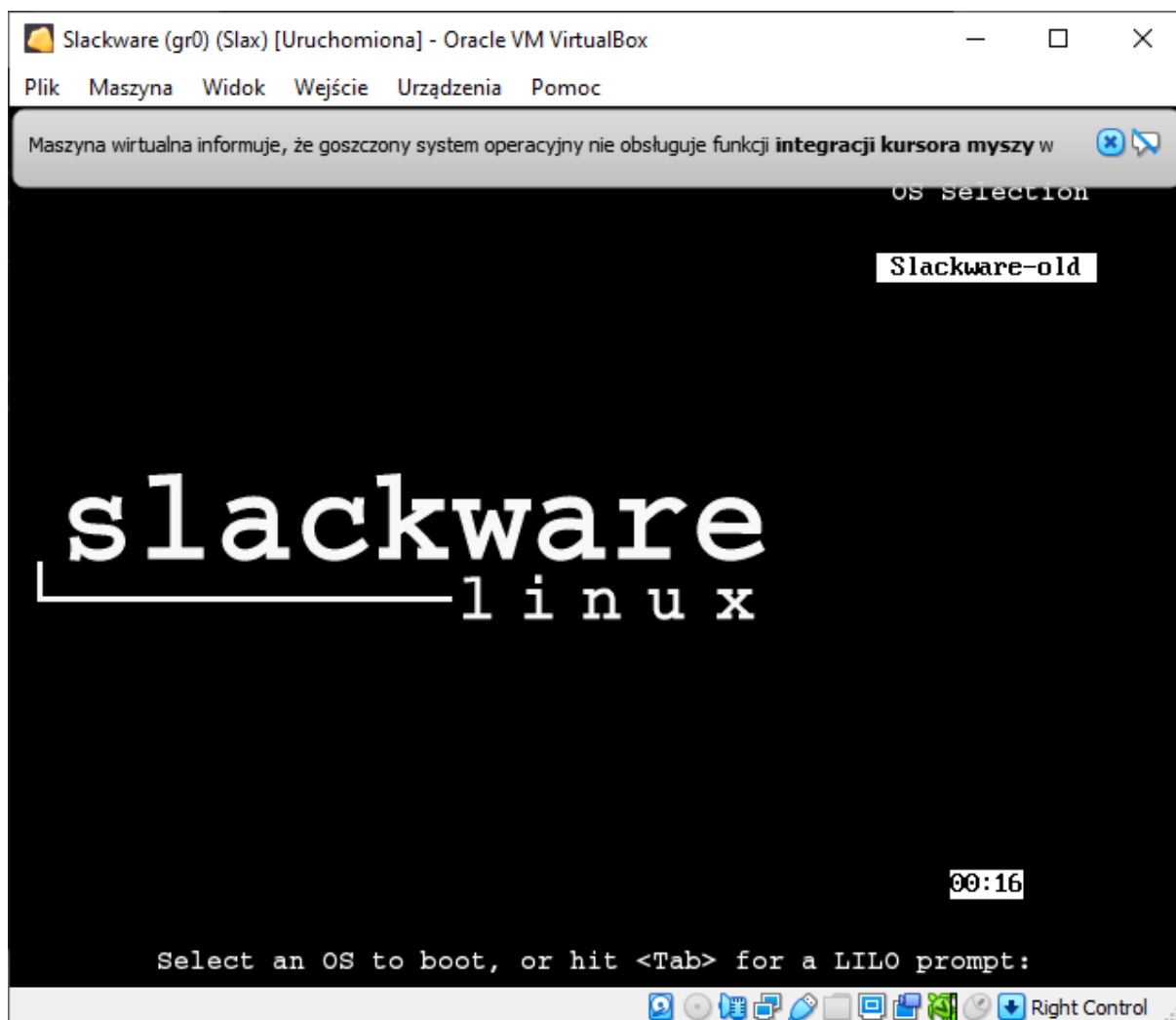
Rysunek 2.12: Edycja lilo.conf



```
OpenSSH SSH client
root@slack:/boot# nano /etc/lilo.conf
root@slack:/boot# 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.
Fatal: Image name, label, or alias is too long: 'Slackware_15.0-old'
root@slack:/boot# nano /etc/lilo.conf
root@slack:/boot# 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-old + *
3 warnings were issued.
root@slack:/boot#
```

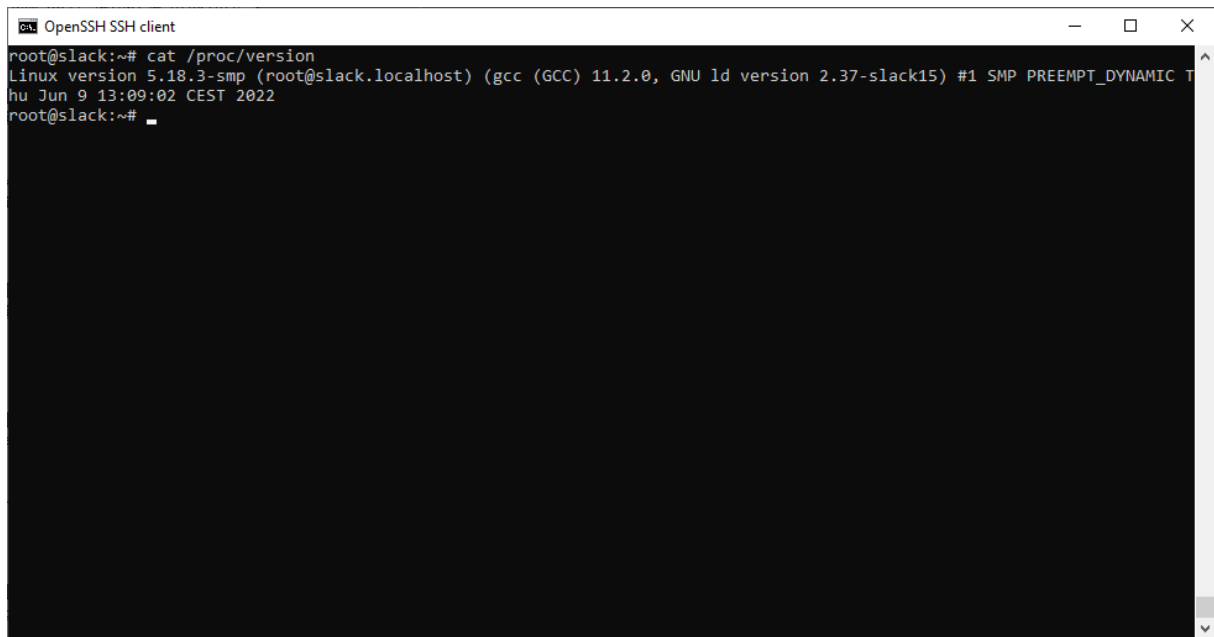
Rysunek 2.13: Uruchomienie lilo

Skoro lilo zostało skonfigurowane, nie pozostaje nic innego jak zrestartować wirtualną maszynę i sprawdzić czy pojawił się odpowiedni wpis.



Rysunek 2.14: Wpis w bootloaderze

Jak pokazuje Rys. 2.14 w bootloaderze możemy wybrać system z labellem ustawionym wcześniej w konfiguracji. Po uruchomieniu tego systemu sprawdzimy jeszcze czy wersja jądra jest odpowiednia.



```
OpenSSH SSH client
root@slack:~# cat /proc/version
Linux version 5.18.3-smp (root@slack.localhost) (gcc (GCC) 11.2.0, GNU ld version 2.37-slack15) #1 SMP PREEMPT_DYNAMIC T
hu Jun 9 13:09:02 CEST 2022
root@slack:~#
```

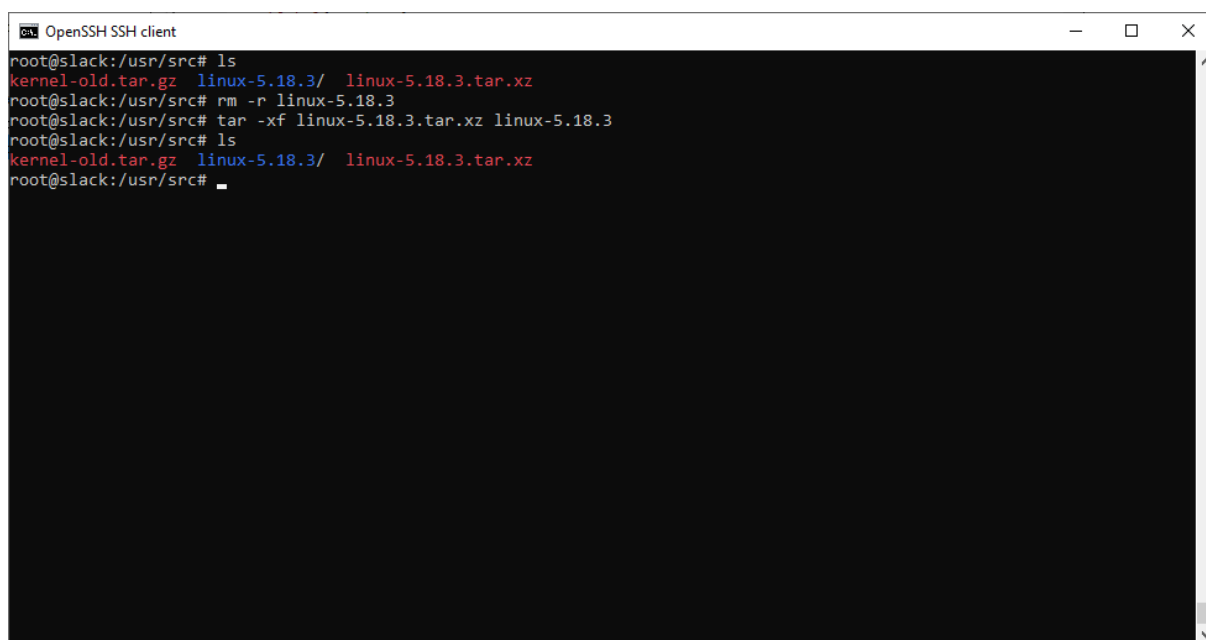
Rysunek 2.15: Wersja jądra po całym procesie

Jak widzimy na Rys. 2.15 wersja jądra to teraz **5.18.3-smp** czyli dokładnie taka jaką kompilowaliśmy, a więc wszystko przebiegło pomyślnie.

Rozdział 3

Kompilacja jądra - new method

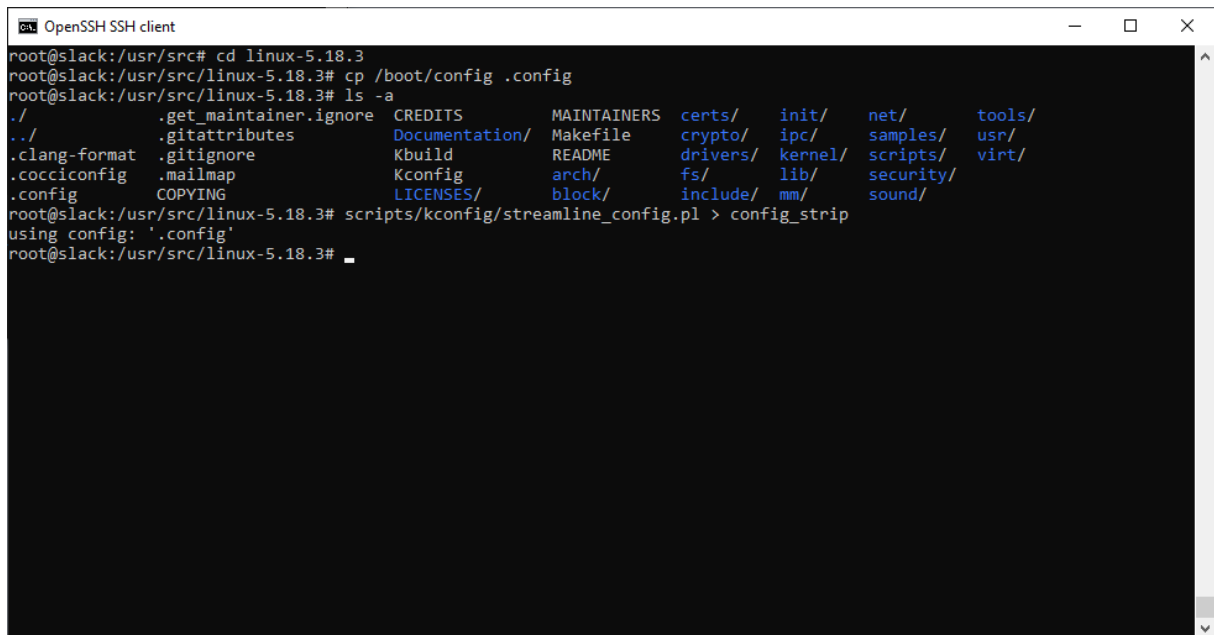
Na początku kompilacji nową metodą usunąłem folder **linux-5.18.3**, który uprzednio został spakowany do **kernel-old.tar.gz** w celu umieszczenia w repozytorium. Później ponownie rozpakowałem pobrane archiwum, tak aby od nowa pracować na czystej wersji (Rys. 3.1)



```
root@slack:/usr/src# ls
kernel-old.tar.gz  linux-5.18.3/  linux-5.18.3.tar.xz
root@slack:/usr/src# rm -r linux-5.18.3
root@slack:/usr/src# tar -xf linux-5.18.3.tar.xz linux-5.18.3
root@slack:/usr/src# ls
kernel-old.tar.gz  linux-5.18.3/  linux-5.18.3.tar.xz
root@slack:/usr/src#
```

Rysunek 3.1: Ponowne rozpakowanie archiwum

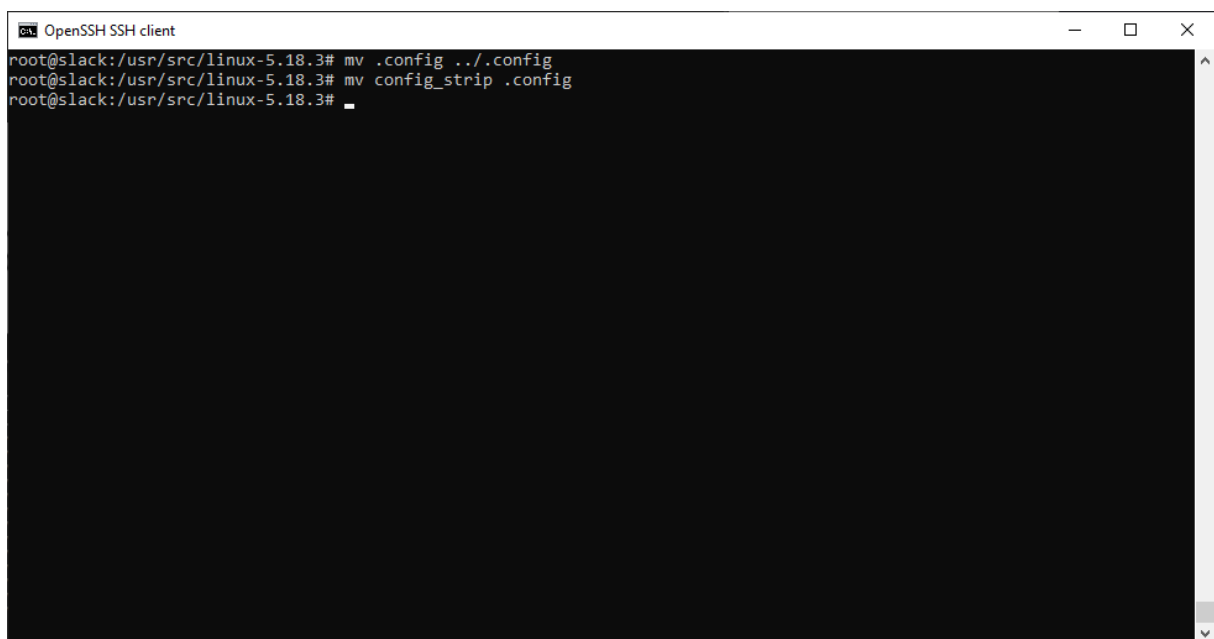
Zgodnie z instrukcjami zawartymi w pliku *streamline-config.pl* kopiuję plik */boot/config* oraz uruchamiam powyższy skrypt przekierowując jego wynik do *config-strip* (Rys. 3.2).



```
OpenSSH SSH client
root@slack:/usr/src# cd linux-5.18.3
root@slack:/usr/src/linux-5.18.3# cp /boot/config .config
root@slack:/usr/src/linux-5.18.3# ls -a
./          .get_maintainer.ignore  CREDITS      MAINTAINERS  certs/      init/      net/      tools/
../         .gitattributes          Documentation/ Makefile      crypto/     ipc/      samples/  usr/
.clang-format .gitignore             Kbuild       README      drivers/    kernel/    scripts/  virt/
.cocciconfig .mailmap               Kconfig      arch/       fs/         lib/      security/
.config     COPYING               LICENSES/     block/      include/    mm/       sound/
root@slack:/usr/src/linux-5.18.3# scripts/kconfig/streamline_config.pl > config_strip
using config: '.config'
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 3.2: Kopiowanie /boot/config oraz skrypt streamline_config.pl

Przenoszę *.config* folder wyżej tak, aby stworzyć sobie kopię zapasową i następnie zastępuję plik *.config* nowo utworzonym plikiem *config_strip* (Rys. 3.3).



```
OpenSSH SSH client
root@slack:/usr/src/linux-5.18.3# mv .config ../.config
root@slack:/usr/src/linux-5.18.3# mv config_strip .config
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 3.3: Podmiana *.config*

Idąc dalej zgodnie z instrukcjami w pliku *streamline_config.pl* uruchamiam komendę *make oldconfig*. Podobnie jak w przypadku starej metody wszystkie ustawienia pozostawiam domyślne. Wynik komendy prezentuje Rys. 3.4.

```
OpenSSH SSH client
Test kstrtou*() 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 uuid 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
Test module loading with 'hello world' module (TEST_LKM) [N/m/?] n
Test module for compilation of bitops operations (TEST_BITOPS) [N/m/?] n
Test module for stress/performance analysis of vmalloc allocator (TEST_VMALLOCC) [N/m/?] n
Test user/kernel boundary protections (TEST_USER_COPY) [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
Test memcat_p() helper function (TEST_MEMCAT_P) [N/m/y/?] 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-5.18.3#
```

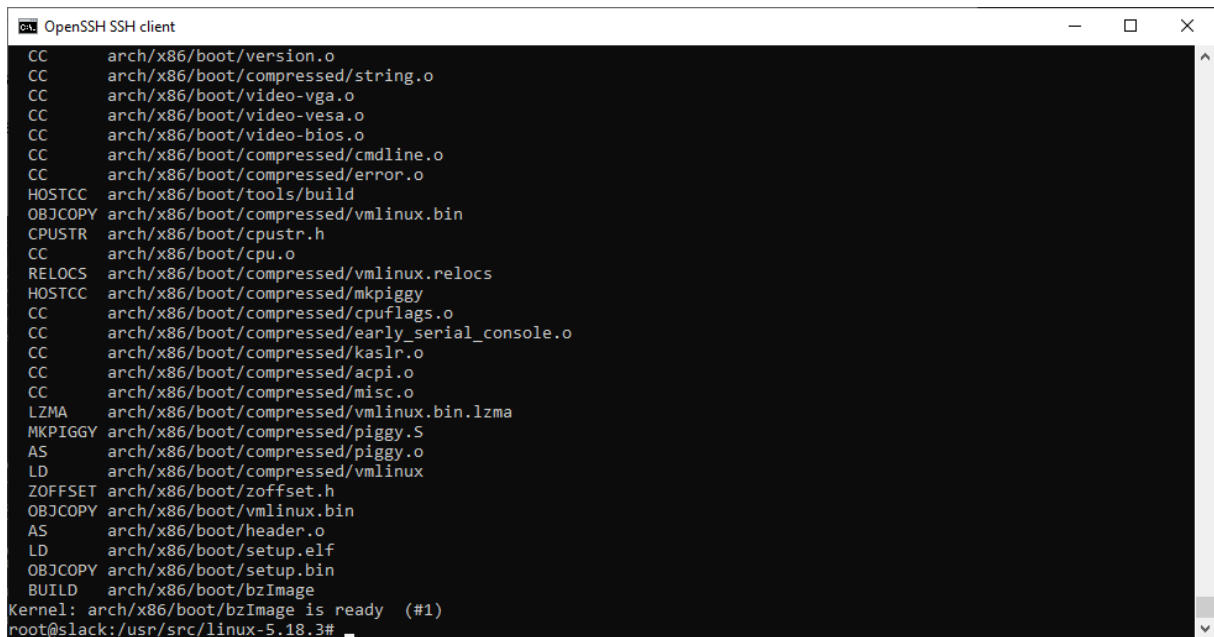
Rysunek 3.4: Wykonanie make oldconfig

Teraz zgodnie z informacją w pliku z instrukcjami nasze jądro jest gotowe do kompilacji. Podobnie jak w starej metodzie uruchomiłem więc polecenie *make -j4 bzImage* (Rys. 3.5).

```
OpenSSH SSH client
# configuration written to .config
#
root@slack:/usr/src/linux-5.18.3#
root@slack:/usr/src/linux-5.18.3#
root@slack:/usr/src/linux-5.18.3#
root@slack:/usr/src/linux-5.18.3#
root@slack:/usr/src/linux-5.18.3# make -j4 bzImage
SYSHDR arch/x86/include/generated/uapi/asm/unistd_32.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_64.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_x32.h
SYSTBL arch/x86/include/generated/asm/syscalls_32.h
WRAP arch/x86/include/generated/uapi/asm/bpf_perf_event.h
WRAP arch/x86/include/generated/uapi/asm/errno.h
WRAP arch/x86/include/generated/uapi/asm/fcntl.h
WRAP arch/x86/include/generated/uapi/asm/ioctl.h
WRAP arch/x86/include/generated/uapi/asm/ioctls.h
WRAP arch/x86/include/generated/uapi/asm/ipcbuf.h
WRAP arch/x86/include/generated/uapi/asm/param.h
WRAP arch/x86/include/generated/uapi/asm/poll.h
WRAP arch/x86/include/generated/uapi/asm/resource.h
WRAP arch/x86/include/generated/uapi/asm/socket.h
WRAP arch/x86/include/generated/uapi/asm/sockios.h
WRAP arch/x86/include/generated/uapi/asm/termios.h
WRAP arch/x86/include/generated/uapi/asm/termios.h
WRAP arch/x86/include/generated/uapi/asm/types.h
HOSTCC arch/x86/tools/relocs_32.o
HOSTCC arch/x86/tools/relocs_64.o
HOSTCC arch/x86/tools/relocs_common.o
HOSTLD arch/x86/tools/relocs
```

Rysunek 3.5: Rozpoczęcie procesu kompilacji

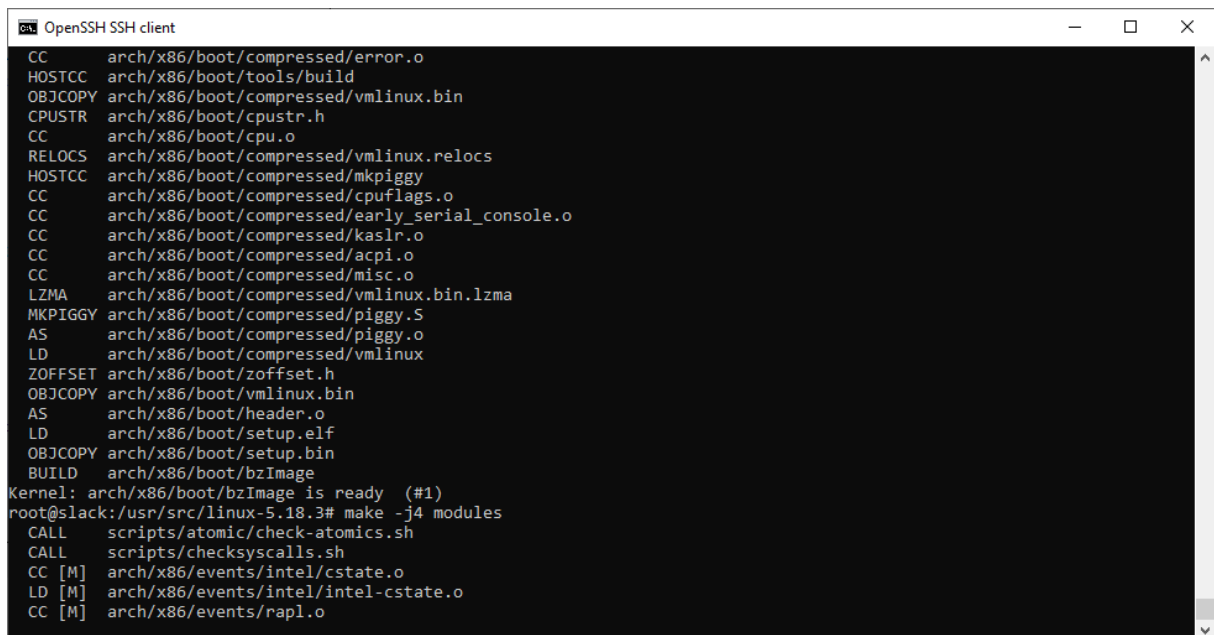
Po kilkunastu minutach proces kompilacji zakończył się pomyślnie (Rys. 3.6).



```
CC      arch/x86/boot/version.o
CC      arch/x86/boot/compressed/string.o
CC      arch/x86/boot/video-vga.o
CC      arch/x86/boot/video-vesa.o
CC      arch/x86/boot/video-bios.o
CC      arch/x86/boot/compressed/cmdline.o
CC      arch/x86/boot/compressed/error.o
HOSTCC  arch/x86/boot/tools/build
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
CPUSTR  arch/x86/boot/cpustr.h
CC      arch/x86/boot/cpu.o
RELOCS  arch/x86/boot/compressed/vmlinux.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/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)
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 3.6: Zakończenie procesu kompilacji

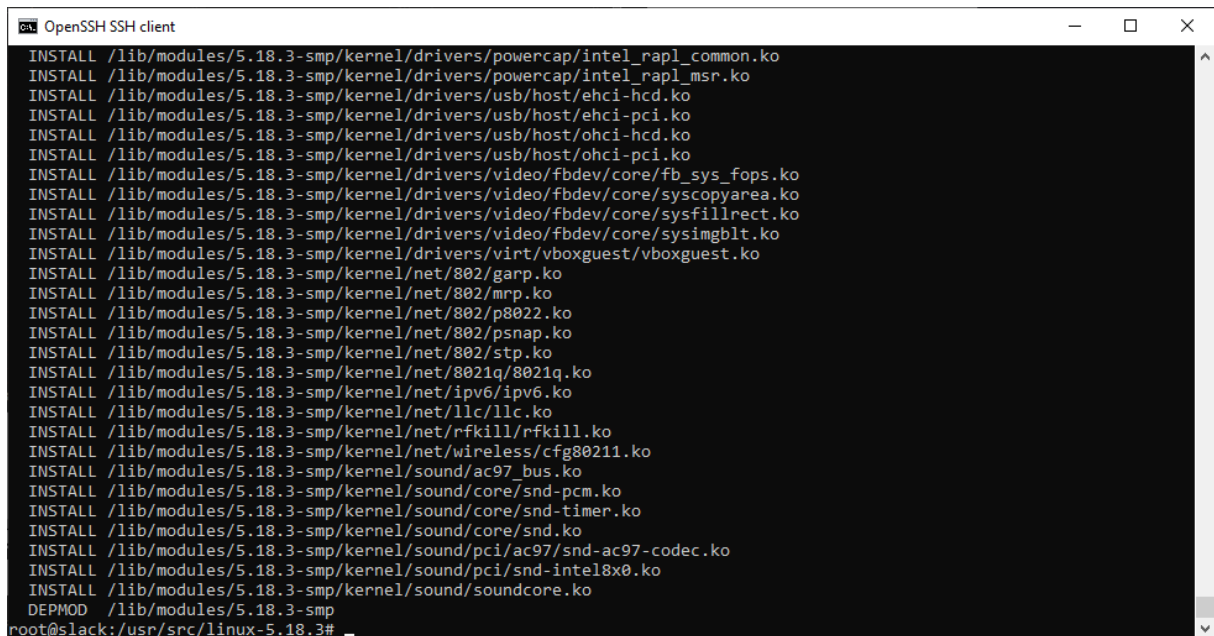
Następnie rozpocząłem proces kompilacji modułów komendą *make -j4 modules* (Rys. 3.7).



```
CC      arch/x86/boot/compressed/error.o
HOSTCC  arch/x86/boot/tools/build
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
CPUSTR  arch/x86/boot/cpustr.h
CC      arch/x86/boot/cpu.o
RELOCS  arch/x86/boot/compressed/vmlinux.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/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)
root@slack:/usr/src/linux-5.18.3# make -j4 modules
CALL    scripts/atomic/check-atomics.sh
CALL    scripts/checksyscalls.sh
CC [M]  arch/x86/events/intel/cstate.o
LD [M]  arch/x86/events/intel-cstate.o
CC [M]  arch/x86/events/rapl.o
```

Rysunek 3.7: Rozpoczęcie procesu kompilacji modułów

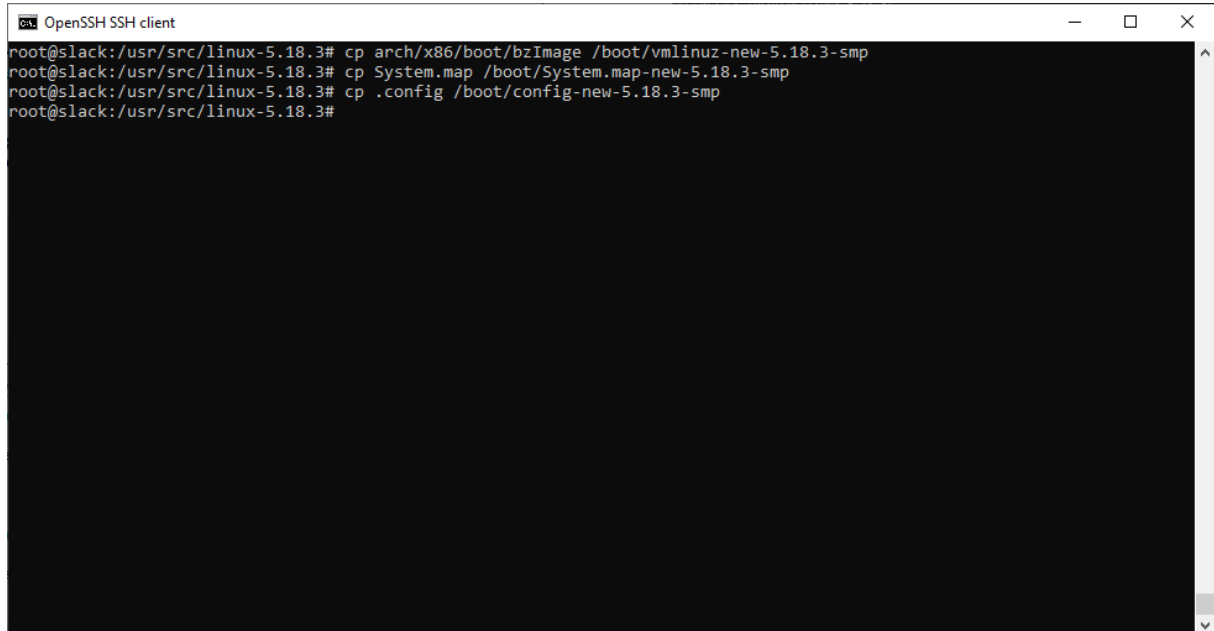
Następnie należy zainstalować moduły komendą *make modules_install*. Wynik komendy prezentuje Rys. 3.8.



```
OpenSSH SSH client
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/powercap/intel_rapl_common.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/powercap/intel_rapl_msr.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/usb/host/ehci-hcd.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/usb/host/ehci-pci.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/usb/host/ohci-hcd.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/usb/host/ohci-pci.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/video/fbdev/core/fb_sys_fops.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/video/fbdev/core/syscopyarea.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/video/fbdev/core/sysfillrect.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/video/fbdev/core/sysimgblt.ko
INSTALL /lib/modules/5.18.3-smp/kernel/drivers/virt/vboxguest/vboxguest.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/802/garp.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/802/mrp.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/802/p8022.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/802/psnap.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/802/stp.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/8021q/8021q.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/ipv6/ipv6.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/llc/llc.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/rfkill/rfkill.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/wireless/cfg80211.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/ac97_bus.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/core/snd-pcm.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/core/snd-timer.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/core/snd.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/pci/ac97/snd-ac97-codec.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/pci/snd-intel8x0.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/soundcore.ko
DEPMOD /lib/modules/5.18.3-smp
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 3.8: Wynik instalacji modułów

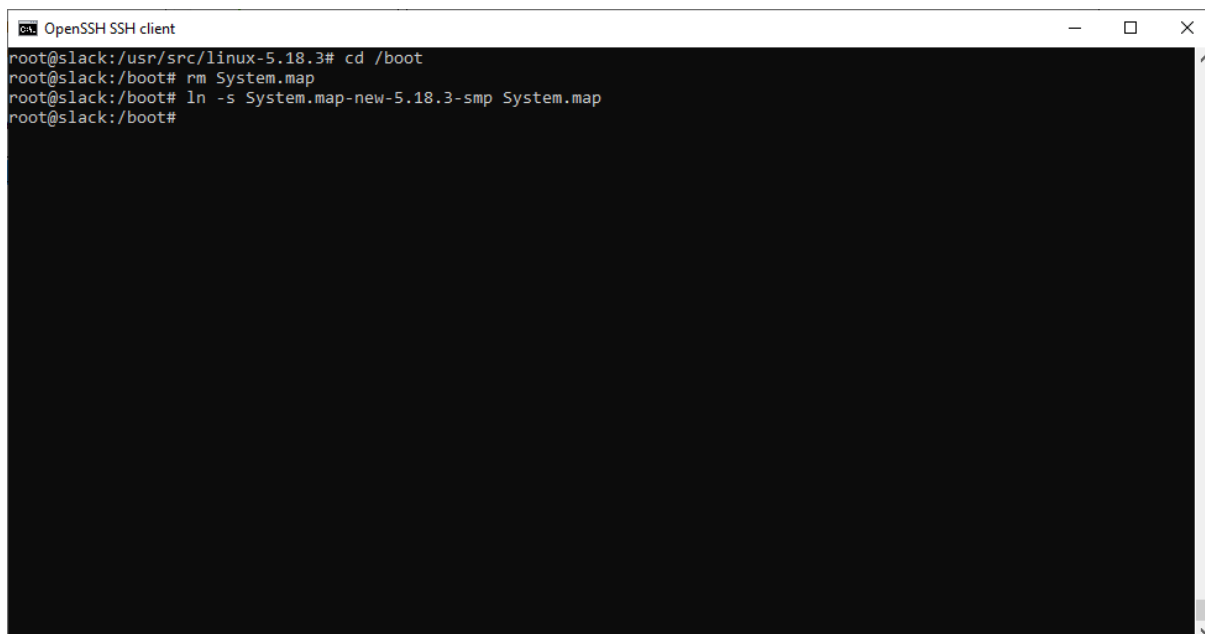
Po zainstalowaniu modułów należy przekopiować niezbędne pliki do katalogu `/boot`, aby mieć możliwość uruchomienia nowego jądra. Tym razem dodałem prefix `new` w celu identyfikacji plików jako utworzonych nową metodą (Rys. 3.9).



```
OpenSSH SSH client
root@slack:/usr/src/linux-5.18.3# cp arch/x86/boot/bzImage /boot/vmlinuz-new-5.18.3-smp
root@slack:/usr/src/linux-5.18.3# cp System.map /boot/System.map-new-5.18.3-smp
root@slack:/usr/src/linux-5.18.3# cp .config /boot/config-new-5.18.3-smp
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 3.9: Kopiowanie do katalogu boot

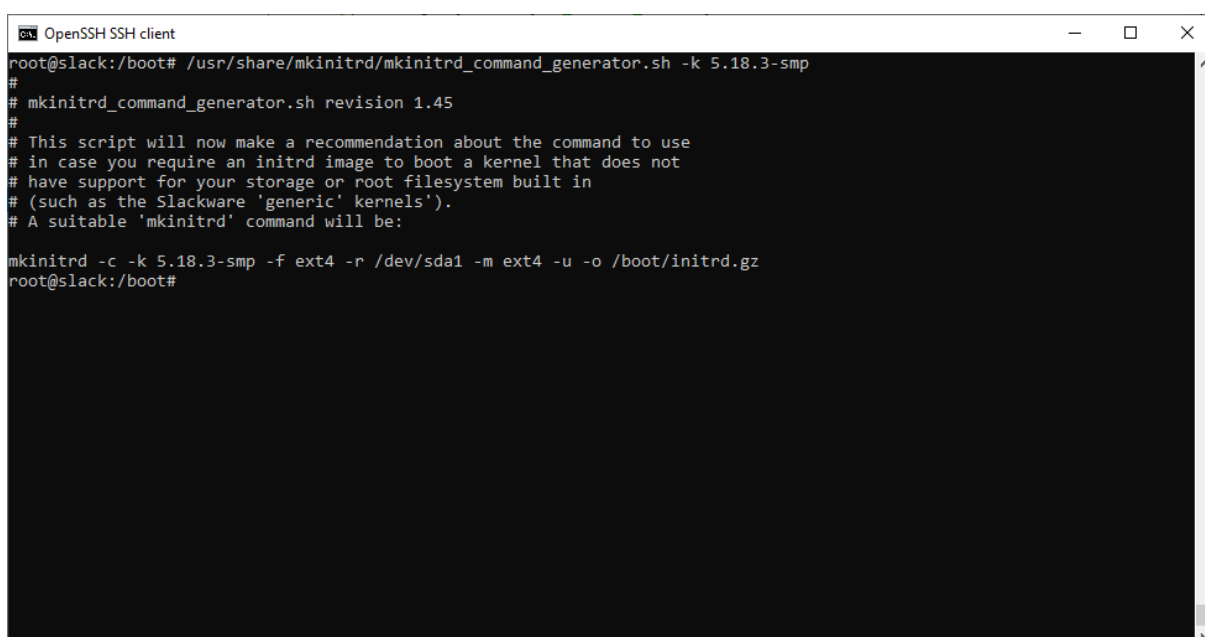
W kolejnym kroku w katalogu `boot` usuwam dotychczasową tablicę symboli `System.map` i tworzę dowiązanie symboliczne do nowego pliku, który przenieśliśmy do tego folderu (Rys. 3.10).



```
OpenSSH SSH client
root@slack:/usr/src/linux-5.18.3# cd /boot
root@slack:/boot# rm System.map
root@slack:/boot# ln -s System.map-new-5.18.3-smp System.map
root@slack:/boot#
```

Rysunek 3.10: Utworzenie nowego dowiązania symbolicznego

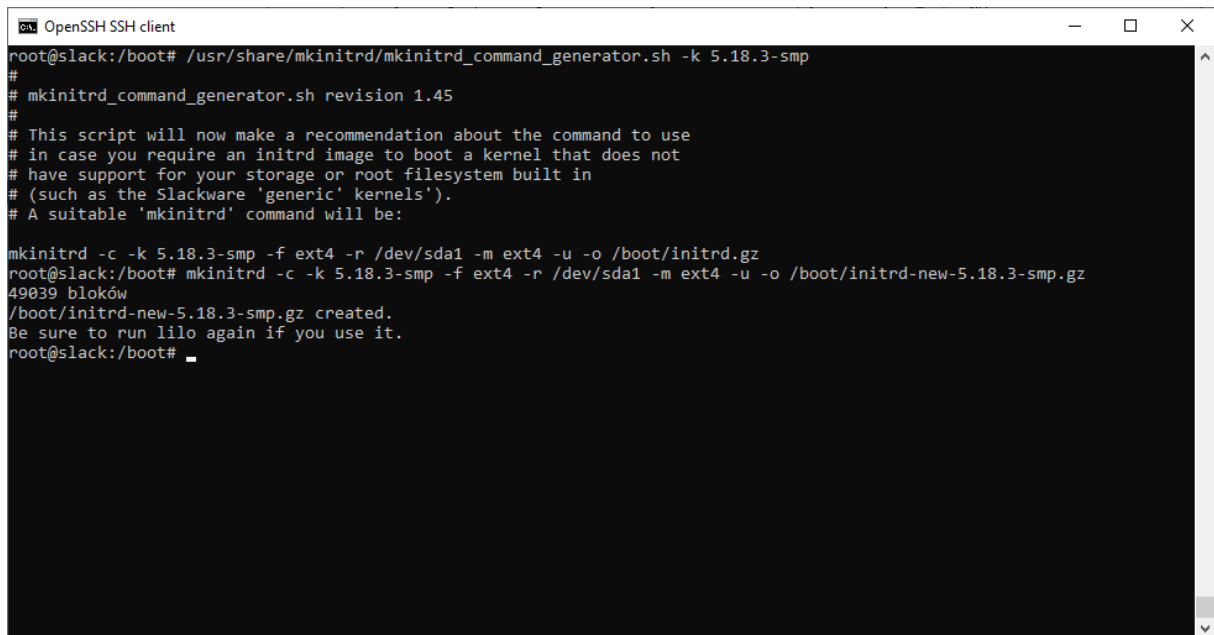
Następnie za pomocą przeznaczonego do tego narzędzia generuję komendę, która pozwoli na utworzenie dysku RAM. Za pomocą opcji *-k* precyzuję wersję jądra (Rys 3.11).



```
OpenSSH SSH client
root@slack:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.18.3-smp
#
# mkinitrd_command_generator.sh revision 1.45
#
# This script will now make a recommendation about the command to use
# in case you require an initrd image to boot a kernel that does not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:
mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot#
```

Rysunek 3.11: Generowanie komendy do utworzenia dysku RAM

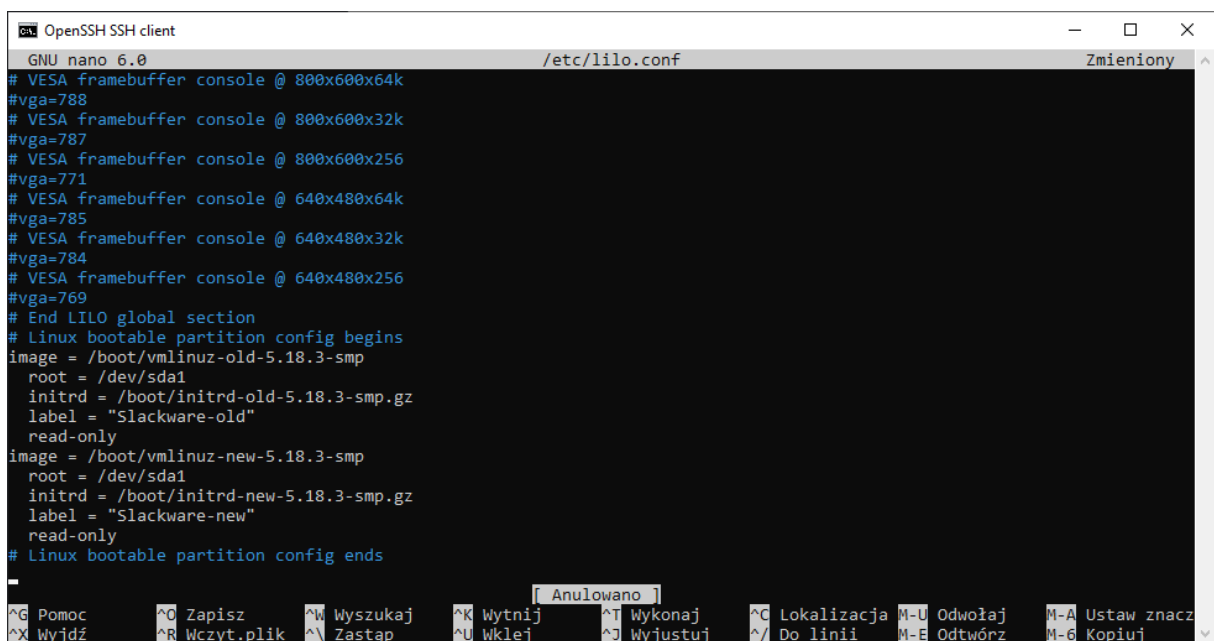
Wygenerowaną komendę należy wywołać, pamiętając o zmianie nazwy generowanego pliku, tak aby odpowiadał naszej wersji jądra. Wynik komendy prezentuje Rys. 3.12.



```
OpenSSH SSH client
root@slack:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.18.3-smp
#
# mkinitrd_command_generator.sh revision 1.45
#
# This script will now make a recommendation about the command to use
# in case you require an initrd image to boot a kernel that does not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:
mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot# mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-new-5.18.3-smp.gz
49039 bloków
/boot/initrd-new-5.18.3-smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot#
```

Rysunek 3.12: Wywołanie komendy do utworzenia dysku RAM

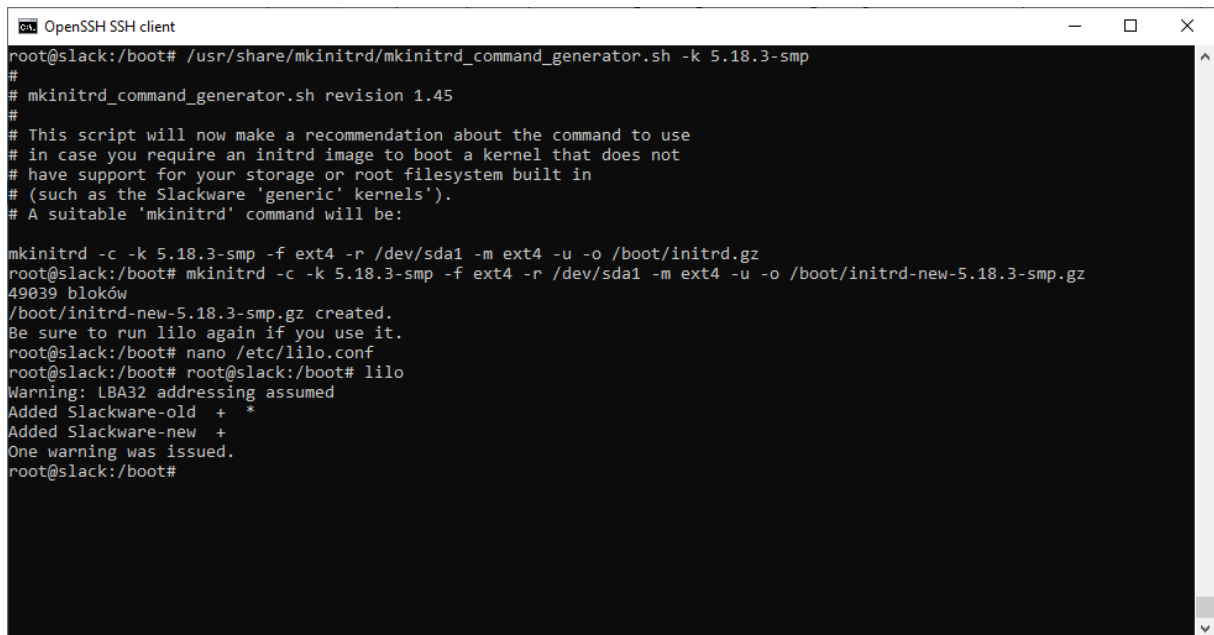
Następnie za pomocą narzędzia nano edytuję plik `/etc/lilo.conf`, tak aby dodać nowy obraz do bootloadera. (Rys. 3.13).



```
GNU nano 6.0 /etc/lilo.conf Zmieniony
# 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-old-5.18.3-smp
  root = /dev/sda1
  initrd = /boot/initrd-old-5.18.3-smp.gz
  label = "Slackware-old"
  read-only
image = /boot/vmlinuz-new-5.18.3-smp
  root = /dev/sda1
  initrd = /boot/initrd-new-5.18.3-smp.gz
  label = "Slackware-new"
  read-only
# Linux bootable partition config ends
-
```

Rysunek 3.13: Edycja lilo.conf

Po konfiguracji uruchomiłem komendę `lilo` i uzyskałem wynik, który prezentuje Rys. 3.14.

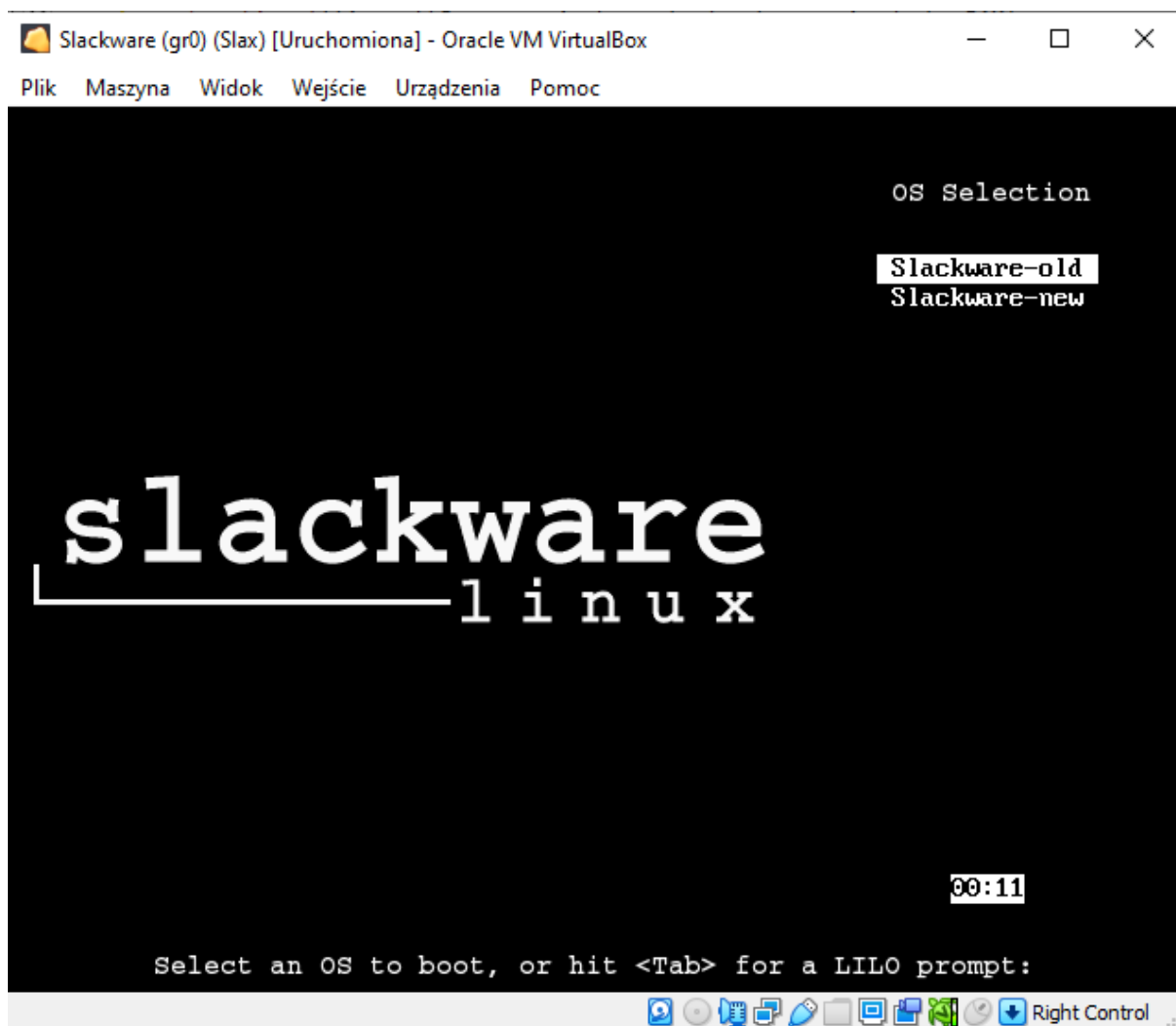


```
OpenSSH SSH client
root@slack:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.18.3-smp
#
# mkinitrd_command_generator.sh revision 1.45
#
# This script will now make a recommendation about the command to use
# in case you require an initrd image to boot a kernel that does not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:
mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot# mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-new-5.18.3-smp.gz
49039 bloków
/boot/initrd-new-5.18.3-smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot# nano /etc/lilo.conf
root@slack:/boot# root@slack:/boot# lilo
Warning: LBA32 addressing assumed
Added Slackware-old + *
Added Slackware-new +
One warning was issued.
root@slack:/boot#
```

Rysunek 3.14: Uruchomienie lilo

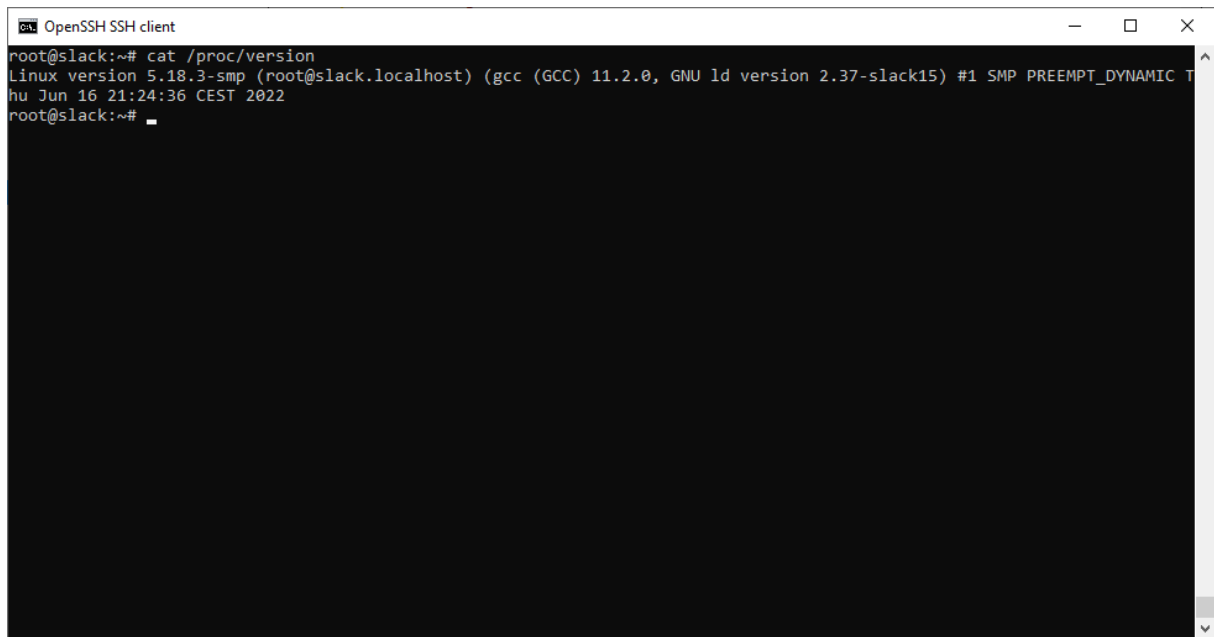
W związku z brakiem błędów pozostaje tylko zrestartować wirtualną maszynę i sprawdzić czy wszystko działa poprawnie.

Po restarcie możemy w bootloaderze wybrać system z labellem ustawionym wcześniej w konfiguracji. (Rys. 3.15).



Rysunek 3.15: Wpisy w bootloaderze

W ostatnim kroku warto sprawdzić czy na nowym systemie wersja jądra jest odpowiednia.



```
OpenSSH SSH client
root@slack:~# cat /proc/version
Linux version 5.18.3-smp (root@slack.localhost) (gcc (GCC) 11.2.0, GNU ld version 2.37-slack15) #1 SMP PREEMPT_DYNAMIC T
hu Jun 16 21:24:36 CEST 2022
root@slack:~#
```

Rysunek 3.16: Wersja jądra po całym procesie

Jak pokazuje Rys. 2.15 wersja jądra to teraz **5.18.3-smp** czyli dokładnie ta, którą kompilowaliśmy, a więc znów wszystko przebiegło pomyślnie.

Rozdział 4

Wnioski

Po przejściu obu metod stwierdzam, że tak naprawdę są to dwie różne metody generowania konfiguracji, ponieważ sam proces kompilacji niczym się nie różni. Obie metody przebiegły u mnie pomyślnie, jednak uważam, że delikatnie bardziej *"user-friendly"* była nowa metoda. Nie do końca rozumiałem parametry ustawiane przy generowaniu konfiguracji, co na szczęście też nie generowało problemów, ponieważ dla przeprowadzanego przez nas procesu wystarczyły domyślne opcje.

Osobiście uważam, że następnym razem wybrałbym nową metodą, jednak gdyby powstała konieczność skorzystania ze starej, nie miałbym z tym problemów.