Kompilacja Kernela Linux

Metoda stara i nowa

Rafał Hrabia
nr indeksu 296583

Informatyka I st.
Uniwersytet Marii Curie-Skłodowskiej
15 czerwca 2022

Spis treści

Rozdział 1

Przygotowanie

W pierwszej kolejności po zalogowaniu się i przejściu do katalogu */usr/src* musimy zdobyć link do paczki z plikami źródłowymi kernela. Z racji tego, że korzystam z wirtualnej maszyny bez interfejsu graficznego posłużę się komputerem, który hostuje maszynę wirtualną. Wchodzimy na stronę i przy użyciu menu kontekstowego kopiujemy do schowka bezpośredni link to pliku(rys. 1.1).

Rysunek 1.1: Kopiowanie linku

Do pobrania pliku jedynym słusznym wyborem będzie komenda *wget*(rys. 1.2).

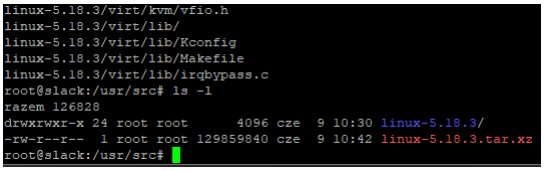
```
root@slack:/usr/src# wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
--2022-06-14 17:24:06--  https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
Translacja cdn.kernel.org (cdn.kernel.org)... 151.101.1.176, 151.101.65.176, 151.101.129.176, ...
Łączenie się z cdn.kernel.org (cdn.kernel.org)|151.101.1.176|: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  1,78MB/s     w 69s

2022-06-14 17:25:15 (1,78 MB/s) - zapisano `linux-5.18.3.tar.xz' [129859840/129859840]

root@slack:/usr/src# ls -l
razem 126824
-rw-r--r-- 1 root root 129859840 cze  9 10:42 linux-5.18.3.tar.xz
root@slack:/usr/src#
```

Rysunek 1.2: Pobieranie kernela

Jako, że pobrany plik jest to plik archiwum używamy na nim komendy *tar -xvpf* (rys 1.3).

Dla przypomnienia (flagi polecenia *tar*):

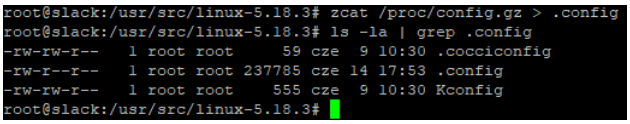
- x – wyodrębnia wymienione pliki
- v – wypisuje nazwy wszystkich plików
- f – określa nazwę pliku archiwum tar
- p - zachowuje ustawienia dostępu do plików



Rysunek 1.3: Rozpakowywanie archiwum

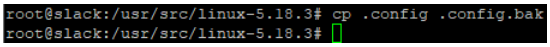
Jak można zauważyć został stworzony nowy folder zawierający pliki z archiwum.

Teraz musimy przekopiować *config* znajdujący się w katalogu */proc* (rys 1.4). Ponieważ plik konfiguracji ma rozszerzenie wskazujące na to, że to archiwum gzip używamy polecenia *zcat*.



Rysunek 1.4: Kopiowanie konfiguracji

Przyda się zrobić kopię zapasową oryginalnego configu. W tym celu kopiujemy plik *.config* do przykładowo pliku *.config.bak* (rys. 1.5).

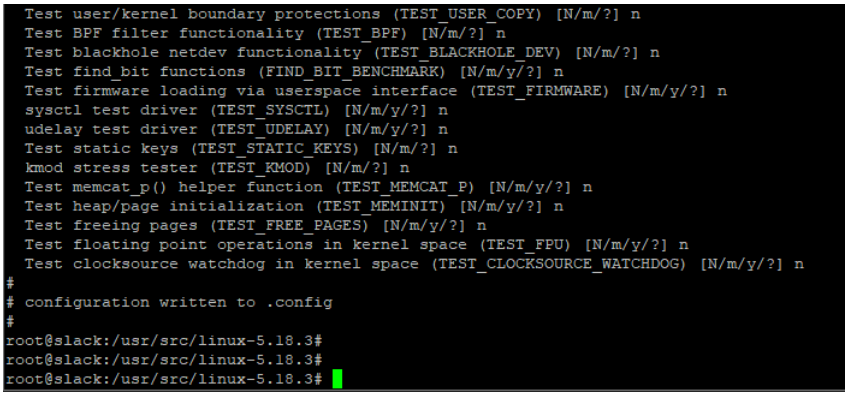


Rysunek 1.5: Kopia zapasowa

Rozdział 2

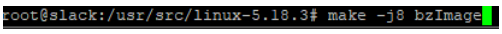
Metoda stara

Przyszła pora na wywołanie polecenia *make localmodconfig*. Po pomyślnym wykonaniu jesteśmy informowani o nadpisaniu konfiguracji do pliku *.config* (rys. 2.1).



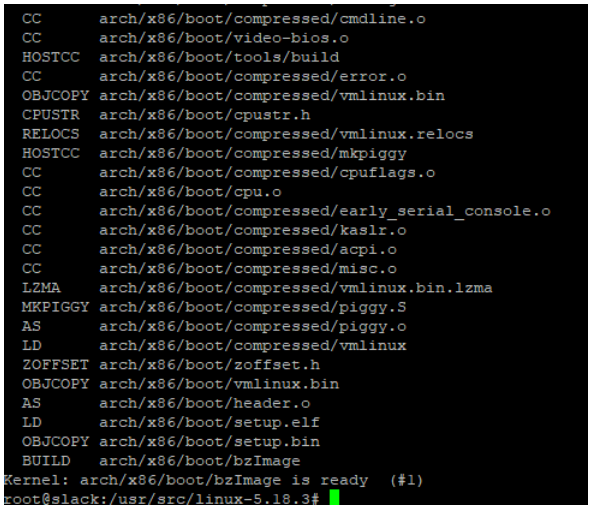
Rysunek 2.1: Wynik polecenia *make localmodconfig*

Nastała wielka chwila - kompilacja jądra. W tym celu trzeba znowu użyć polecenia *make*. Tym razem użyjemy parametru *-j< liczba >* co pozwoli nam uruchomić kompilację kernela na wielu rdzeniach(lub wątkach?). W moim przypadku użyję liczby 8.



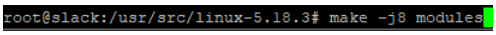
Rysunek 2.2: Kompilowanie kernela

Kompilacja kernela przebiegła w zawrotnym czasie 4 minut i 45 sekund (mniej więcej).



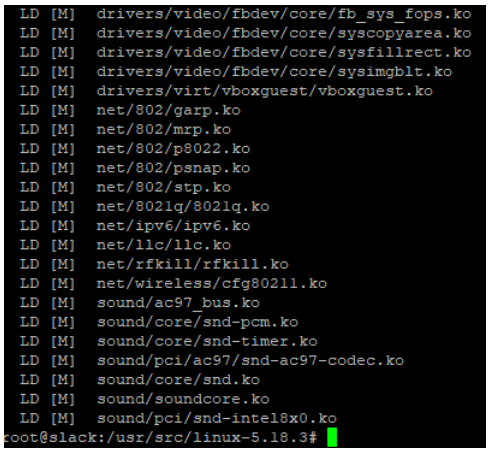
Rysunek 2.3: Kompilowanie kernela - wynik

Kompilację modułów zleca się również komendą make, ale zamieniamy parametr pozycyjny *bzImage* na *modules*.



Rysunek 2.4: Kompilowanie modułów

Budowanie modułów zakończyło się zaskakująco szybko - w niecałe 36 sekund (tak, stałem ze stoperem w ręce). Pytanie czy tak powinno być? Czy to może zasługa Intel Core i9? Podejrzewam, że "stara metoda" ograniczyła ilość modułów do niezbędnego minimum. Przyjmijmy więc, że wszystko jest w porządku i kontynuujmy.



Rysunek 2.5: Kompilowanie modułów - wynik

Instalacja modułów nie wprowadza nic nowego. Instalujemy moduły przy użyciu *make*.

```
root@slack:/usr/src/linux-5.18.3# make -j8 modules_install
```

Rysunek 2.6: Instalowanie modułów

Bardzo szybkie 3 sekundy później (znowu mierzyłem) proces jest zakończony.

```
  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/wireless/cfg80211.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/net/rfkill/rfkill.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/sound/ac97_bus.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/net/llc/llc.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#
```

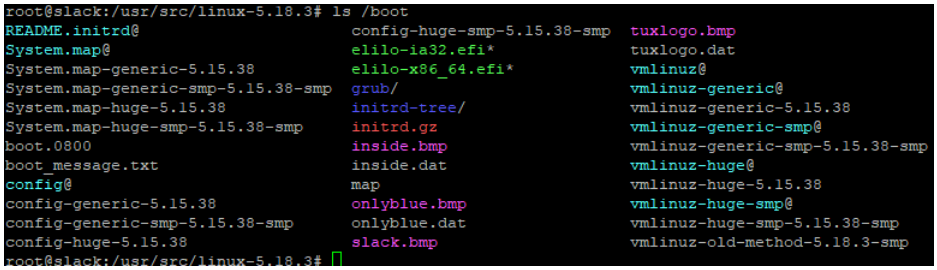
Rysunek 2.7: Instalowanie modułów - wynik

Możemy teraz przystąpić do kopiowania kernela do katalogu */boot*.

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

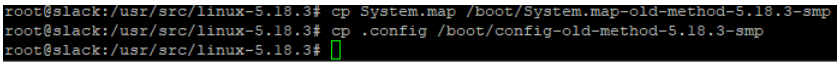
Rysunek 2.8: Kopiowanie bzImage do katalogu /boot

Plik jądra został z sukcesem przekopiowany do katalogu */boot*.



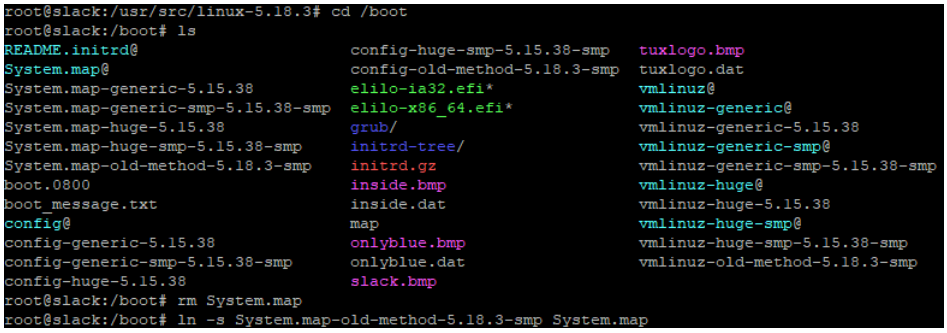
Rysunek 2.9: Kopiowanie bzImage do katalogu /boot - wynik

Musimy jeszcze przekopiować konfigurację i mapę systemu.



Rysunek 2.10: Kopiowanie configu i mapy symboli do katalogu /boot

Teraz możemy usunąć oryginalny plik *System.map* i podłączyć skopiowany korzystając z linku symbolicznego.



Rysunek 2.11: Utworzenie linku symbolicznego do skopiowanej mapy symboli.

Generujemy dysk startowy. Do wygenerowania polecenia używamy skryptu */usr/share/mkinitrd/mkinitrd_command_generator.sh*. Następnie wykonujemy wygenerowane polecenie.

```
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-old-method-5.18.3-smp.gz
49031 bloków
/boot/initrd-old-method-5.18.3-smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot#
```

Rysunek 2.12: Generujemy dysk ram

Pozostała nam już tylko konfiguracja lilo.

```
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 15.0"
  read-only

image = /boot/vmlinuz-old-method-5.18.3-smp
  root = /dev/sda1
  initrd = /boot/initrd-old-method-5.18.3-smp.gz
  label = "Old Method"
  read-only
# Linux bootable partition config ends
```

Rysunek 2.13: Konfiguracja lilo

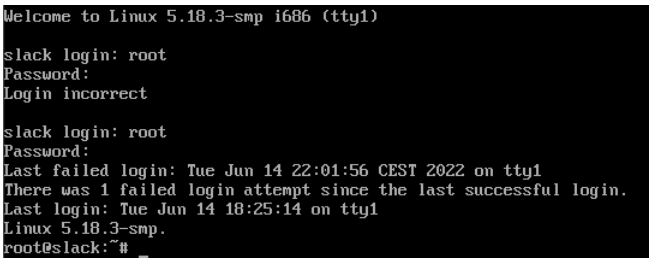
```
root@slack:/boot# lilo
Warning: LBA32 addressing assumed
Added Slackware_15.0  *
Added Old_Method  +
One warning was issued.
root@slack:/boot#
```

Rysunek 2.14: Aktualizacja lilo

Po uruchomieniu systemu widzimy nową pozycję "Old Method".

Rysunek 2.15: Widok Lilo po uruchomieniu maszyny wirtualnej

Po chwili system się uruchamia i po zalogowaniu(lub nawet przed zalogowaniem) widzimy, że zmieniła się wersja jądra systemu.



Rysunek 2.16: Logowanie na root'a na nowym kernelu

Słowami zakończenia - nie obyło się bez błędów. Generując dysk ram za pierwszym razem przypadkowo nadpisałem oryginalny plik initrd.gz. Udało mi się sytuację prawdopodobnie uratować przekopiowując plik initrd.gz z poprzedniego snapshota, który utworzyłem, na maszynę, na której znajduje się Virtual Box, a następnie z powrotem na aktualną wersję systemu. Pomimo starań przy próbie odpalenia nowego kernela ten po długim czasie bootowania zaczął sypać błędami. Możliwe też, że to przez to, że przy tworzeniu dysku ram zapomniałem o dodaniu *.gz* na końcu (lilo skonfigurowało się poprawnie i po uruchomieniu była widoczna nowa pozycja). Zauważyłem to już po fakcie. Stwierdziłem, że spróbuję jeszcze raz zaczynając od momentu kompilowania jądra. Za drugim razem poszło gładko i system wystartował. Nie umieszczam jeszcze raz zrzutów ekranu, gdzie treść w pierwszym i drugim podejściu była identyczna, żeby się nie powtarzać.

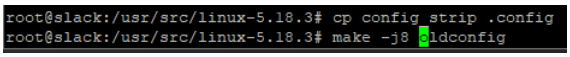
Rozdział 3

Metoda nowa

Pracę nad nową metodą zaczynamy od przygotowania plików źródłowych i skopiowania configu. Teraz możemy stworzyć konfigurację systemu nową metodą.

Rysunek 3.1: Utworzenie configu nową metodą

Tak utworzony config-strip przekopiowujemy do .config i wywołujemy *make oldconfig* zgodnie z instrukcją w skrypcie z poprzedniego zdjęcia.



Rysunek 3.2: Zmiana configu

Teraz trzeba klikać Enter (i to dużo razy). W końcu gra w nie-czytanie-i-klikanie-enter się kończy i musimy przejść do następnego kroku.

```
  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_VMALLOC) [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#
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 3.3: Maraton klikania Enter

Kompilujemy jądro.

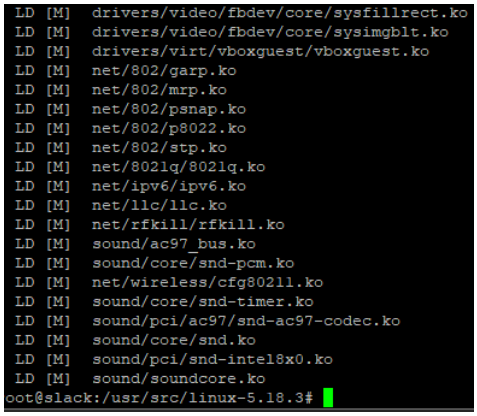
```
root@slack:/usr/src/linux-5.18.3# make -j8 bzImage
```

Rysunek 3.4: Kompilacja jądra

```
  VOFFSET arch/x86/boot/compressed/../voffset.h
  CC      arch/x86/boot/compressed/string.o
  CC      arch/x86/boot/compressed/cmdline.o
  CC      arch/x86/boot/compressed/error.o
  OBJCOPY arch/x86/boot/compressed/vmlinux.bin
  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#
```

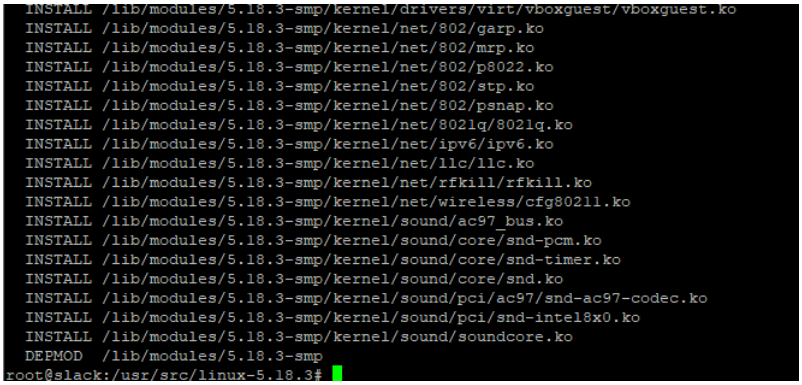
Rysunek 3.5: Kompilacja jądra - wynik

Przechodzimy do kompilacji modułów.



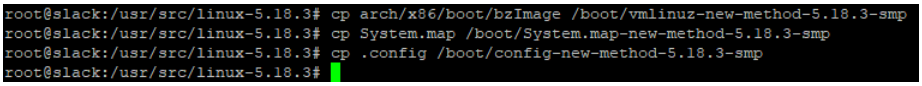
Rysunek 3.6: Kompilacja modułów - wynik

A następnie do instalacji modułów.



Rysunek 3.7: Instalacja modułów - wynik

Kopiujemy niezbędne pliki.



Rysunek 3.8: Kopiowanie plików

Tworzymy link symboliczny do System.map.

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

Rysunek 3.9: Utworzenie linku symbolicznego

Tworzymy dysk ram.

```
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-method-5.18.3-smp
.gz
49031 bloków
/boot/initrd-new-method-5.18.3-smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot#
```

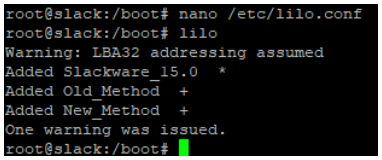
Rysunek 3.10: Stworzenie dysku ram

Konfigurujemy lilo.

```
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 15.0"
  read-only

image = /boot/vmlinuz-old-method-5.18.3-smp
  root = /dev/sda1
  initrd = /boot/initrd-old-method-5.18.3-smp.gz
  label = "Old Method"
  read-only

image = /boot/vmlinuz-new-method-5.18.3-smp
  root = /dev/sda1
  initrd = /boot/initrd-new-method-5.18.3-smp.gz
  label = "New Method"
  read-only
```

Rysunek 3.11: /etc/lilo.conf

I ponownie aktualizujemy

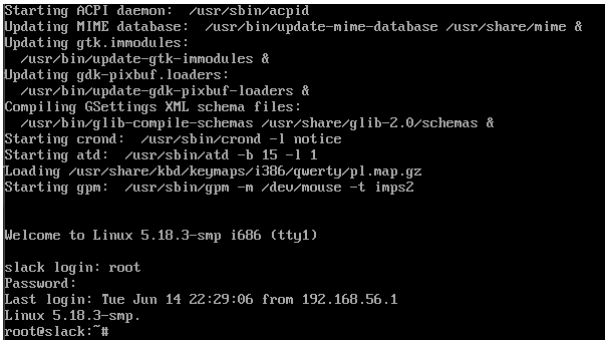


Rysunek 3.12: Wynik komendy lilo

Po ponownym uruchomieniu maszyny mamy kolejną pozycję do wyboru.

Rysunek 3.13: Wynik komendy lilo

Widzimy, że zmieniła się wersja jądra.



Rysunek 3.14: Logowanie na root'a

Rozdział 4

Wnioski

Jak można się przekonać wykonując ćwiczenie - kompilacja kernela nie jest trudna, a jest nawet przyjemna. Przede wszystkim trzeba uważać na błędy ludzkie, takie jak złe nazwanie pliku i wykasowanie lub nadpisanie czegoś przez przypadek. Zdarza się. Oczywiście lekcja jest taka, że backupów nigdy za wiele.