Filip Krasiński

Kompilacja jądra

12.06.2022

Kopiujemy link do pobrania jądra ze strony kernel.org i następnie za pomocą komendy ***wget*** pobieramy jądro do folder ***/usr/src***

```
root@localhost:/usr/src# wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
--2022-06-12 17:19:07--  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  34,9MB/s     w 3,7s

2022-06-12 17:19:11 (33,6 MB/s) - zapisano `linux-5.18.3.tar.xz' [129859840/129859840]

root@localhost:/usr/src#
```

Rysunek 1. Pobieranie jądra za pomocą wget.

1. Kompilacja przy wykorzystaniu starej metody

1. Wypakowujemy jądro za pomocą komendy

   tar -xvpf linux-5.18.3.tar.xz

```
linux-5.18.3/virt/kvm/irqchip.c
linux-5.18.3/virt/kvm/kvm_main.c
linux-5.18.3/virt/kvm/kvm_mm.h
linux-5.18.3/virt/kvm/pfncache.c
linux-5.18.3/virt/kvm/vfio.c
linux-5.18.3/virt/kvm/vfio.h
linux-5.18.3/virt/lib/
linux-5.18.3/virt/lib/Kconfig
linux-5.18.3/virt/lib/Makefile
linux-5.18.3/virt/lib/irqbypass.c
root@localhost:/usr/src#
```

Rysunek 1.1. Wypakowywanie przy użyciu tar.

2. Przechodzimy do folderu z pobranym jądrem i kopiujemy konfigurację

```
root@localhost:/usr/src# cd linux-5.18.3
root@localhost:/usr/src/linux-5.18.3# zcat /proc/config.gz > .config
root@localhost:/usr/src/linux-5.18.3#
```

Rysunek 1.2. Kopiowanie konfiguracji.

3. Za pomocą komendy ***make localmodconfig*** generujemy konfigurację z używanymi modułami

```
root@localhost:/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: '.config'
```

Rysunek 1.3. Generacja konfiguracji przy użyciu make localmodconfig.

4. Następnie zostaniemy poproszeni o ustawienie pewnych parametrów, w naszym przypadku pozostawiamy wszystkie parametry jako domyślne przy pomocy entera

```
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 1.4. Ustawienia konfiguracji.

```
  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@localhost:/usr/src/linux-5.18.3#
```

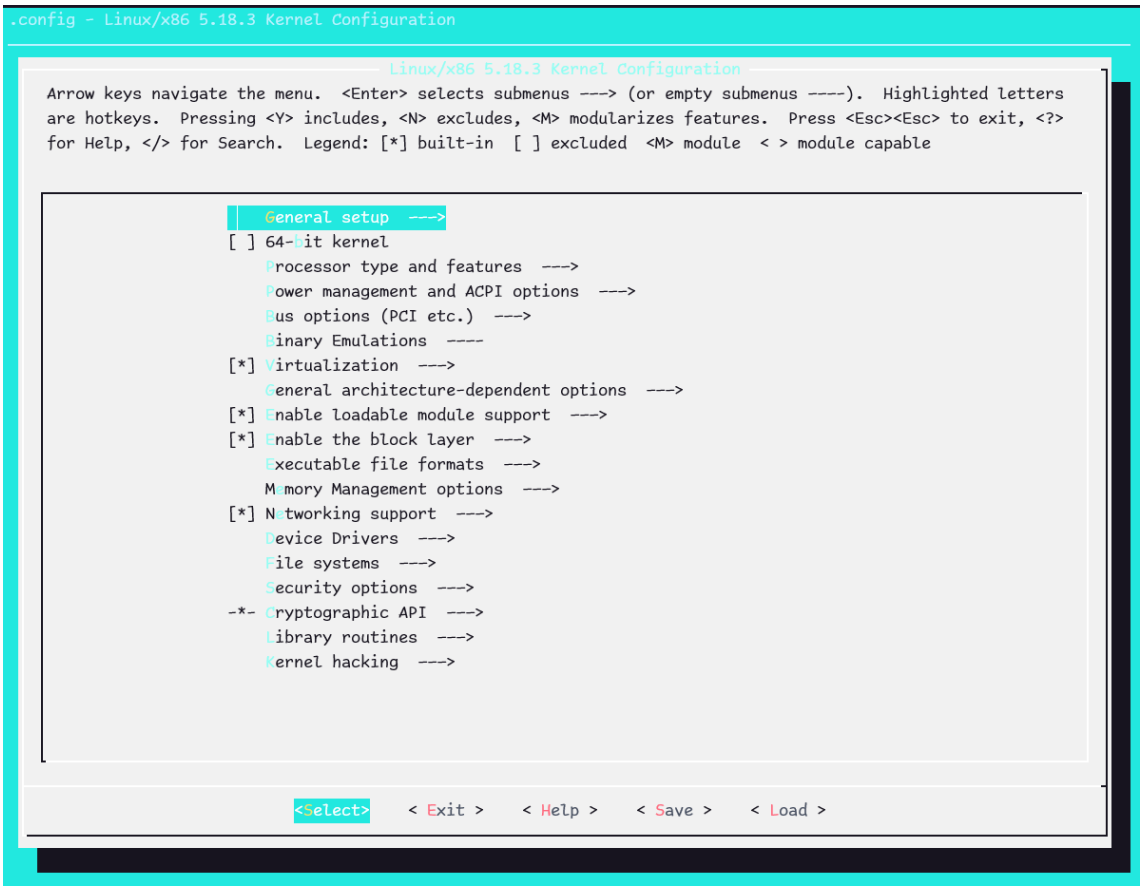
Rysunek 1.5. Zakończenie konfiguracji.

5. Teraz możemy sprawdzić nasze załadowane moduły za pomocą komendy ***lsmod***

```
root@localhost:/usr/src/linux-5.18.3# lsmod
Module                  Size  Used by
vboxvideo              24576  0
drm_vram_helper        20480  1 vboxvideo
drm_ttm_helper         16384  2 drm_vram_helper,vboxvideo
cfg80211              770048  0
8021q                  28672  0
garp                   16384  1 8021q
mrp                    20480  1 8021q
stp                    16384  1 garp
llc                    16384  2 garp,stp
rfkill                 24576  1 cfg80211
ipv6                  458752  20
intel_rapl_msr         20480  0
joydev                 20480  0
intel_rapl_common      24576  1 intel_rapl_msr
vmwgfx                270336  2
ttm                    61440  3 drm_vram_helper,drm_ttm_helper,vmwgfx
drm_kms_helper        237568  3 drm_vram_helper,vboxvideo,vmwgfx
rapl                   20480  0
snd_intel8x0           36864  0
intel_cstate           20480  0
drm                   475136  8 drm_vram_helper,drm_ttm_helper,vboxvideo,vmwgfx,ttm,drm_kms_helper
snd_ac97_codec        122880  1 snd_intel8x0
ohci_pci               16384  0
evdev                  20480  13
ehci_pci               16384  0
snd_pcm               102400  2 snd_ac97_codec,snd_intel8x0
fb_sys_fops            16384  1 drm_kms_helper
psmouse               126976  0
intel_agp              16384  0
snd_timer              32768  1 snd_pcm
```

Rysunek 1.6. Sprawdzenie modułów za pomocą lsmod.

6. Sprawdzamy konfiguracje jądra przy użyciu komendy ***make menuconfig***



Rysunek 1.7. Okno make menuconfig.

```
root@localhost:/usr/src/linux-5.18.3# make menuconfig
  UPD     scripts/kconfig/mconf-cfg
  HOSTCC  scripts/kconfig/mconf.o
  HOSTCC  scripts/kconfig/lxdialog/checklist.o
  HOSTCC  scripts/kconfig/lxdialog/inputbox.o
  HOSTCC  scripts/kconfig/lxdialog/menubox.o
  HOSTCC  scripts/kconfig/lxdialog/textbox.o
  HOSTCC  scripts/kconfig/lxdialog/util.o
  HOSTCC  scripts/kconfig/lxdialog/yesno.o
  HOSTLD  scripts/kconfig/mconf


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

root@localhost:/usr/src/linux-5.18.3#
```

Rysunek 1.8. Zakończenie make menuconfig.

7. Ostatnim krokiem konfiguracji jest zbudowanie pliku konfiguracyjnego jądra

```
root@localhost:/usr/src/linux-5.18.3# make olddefconfig
#
# No change to .config
#
root@localhost:/usr/src/linux-5.18.3#
```

Rysunek 1.9. Budowa pliku konfiguracyjnego za pomocą ***make olddefconfig.***

8. Następnym krokiem jest kompilacja obrazu jądra za pomocą komendy ***make bzImage*** – argument -j odpowiada za to ile rdzeni zostanie wykorzystanych w procesie budowy, w tym przypadku jest to liczba 6 rdzeni. Dodatkowo za pomocą komendy ***time*** zmierzę czas kompilacji jądra aby porównać starą i nową metodę pod kątem czasowym.

```
root@localhost:/usr/src/linux-5.18.3# time make -j6 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/ioctls.h
  WRAP    arch/x86/include/generated/uapi/asm/ipcbuf.h
  WRAP    arch/x86/include/generated/uapi/asm/ioctl.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
```

Rysunek 1.10. Proces budowy obrazu jądra.

```
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)

real    11m11,172s
user    22m22,825s
sys     8m27,272s
root@localhost:/usr/src/linux-5.18.3#
```

Rysunek 1.11. Zakończenie procesu budowy obrazu jądra.

9. Teraz możemy zbudować moduły jądra, tak jak powyżej, argument -j odpowiada za ilość rdzeni wykorzystanych do budowy.

```
root@localhost:/usr/src/linux-5.18.3# time make -j6 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
```

Rysunek 1.12. Budowa modułów.

```
  LD [M]  sound/core/snd-timer.ko
  LD [M]  sound/core/snd.ko
  LD [M]  sound/pci/ac97/snd-ac97-codec.ko
  LD [M]  sound/pci/snd-intel8x0.ko
  LD [M]  sound/soundcore.ko

real    1m7,033s
user    2m47,856s
sys     1m2,768s
root@localhost:/usr/src/linux-5.18.3#
```

Rysunek 1.13. Zakończenie budowy modułów.

10. Instalujemy moduły za pomocą ***make modules_install***

```
root@localhost:/usr/src/linux-5.18.3# make 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/ac.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/button.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/video.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/block/loop.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/char/agp/agpgart.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/drm_vram_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/vboxvideo/vboxvideo.ko
```

Rysunek 1.14. Instalacja modułów.

11. Sprawdzamy zainstalowane moduły

```
root@localhost:/usr/src/linux-5.18.3# ls /lib/modules/5.18.3-smp
build@         modules.alias.bin          modules.builtin.bin      modules.dep.bin  modules.softdep      source@
kernel/        modules.builtin            modules.builtin.modinfo  modules.devname  modules.symbols
modules.alias  modules.builtin.alias.bin  modules.dep              modules.order    modules.symbols.bin
root@localhost:/usr/src/linux-5.18.3#
```

Rysunek 1.15. Spis zainstalowanych modułów.

12. Gdy już wszystko zbudowaliśmy kopiujemy pliki jądra do systemu
    - Kopiowanie obrazu jądra

      cp arch/x86/boot/bzImage /boot/vmlinuz-old-5.18.3-smp

    - Kopiowanie tablicy symboli

      cp System.map /boot/System.map-old-5.18.3-smp

    - Kopiowanie pliku konfiguracyjnego

      cp .config /boot/config-old-5.18.3-smp

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

Rysunek 1.16. Kopiowanie plików jądra.

13. Tworzymy link symboliczny dla tablicy symboli jądra

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

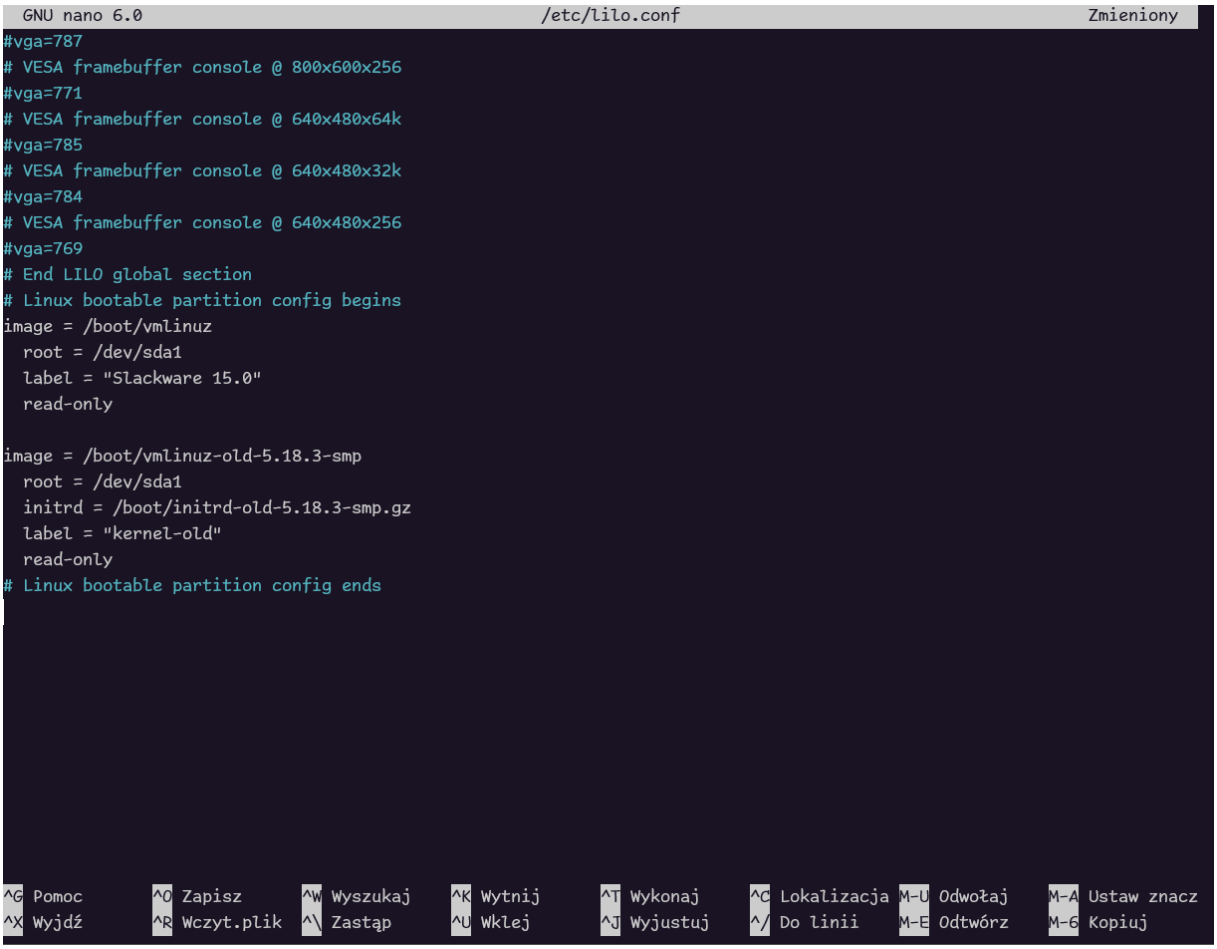
Rysunek 1.17. Tworzenie linku dla tablicy symboli.

14. Tworzymy dysk RAM poprzez wywołanie skryptu generującego dla nas komendę do wykonania, podmieniamy tylko nazwę dysku w wygenerowanej komendzie

```
root@localhost:/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@localhost:/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@localhost:/boot#
```

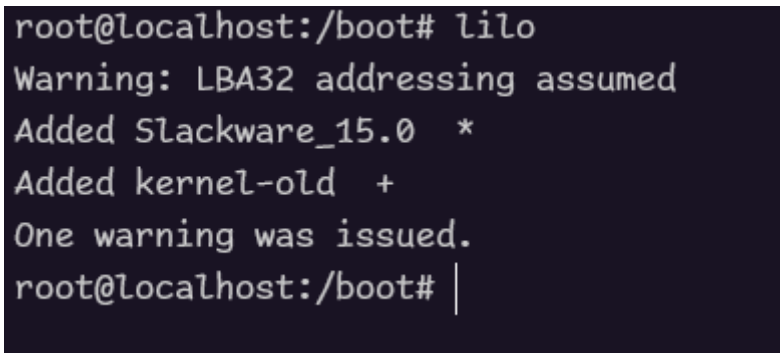
Rysunek 1.18. Tworzenie dysku RAM.

15. Jako jeden z ostatnich kroków dodajemy wpisz do pliku konfiguracyjnego **lilo** za pomocą komendy ***nano /etc/lilo.conf***



Rysunek 1.19. Modyfikacja pliku ***/etc/lilo.conf***

16. Wywołujemy komendę ***lilo*** aby sprawdzić naszą konfigurację

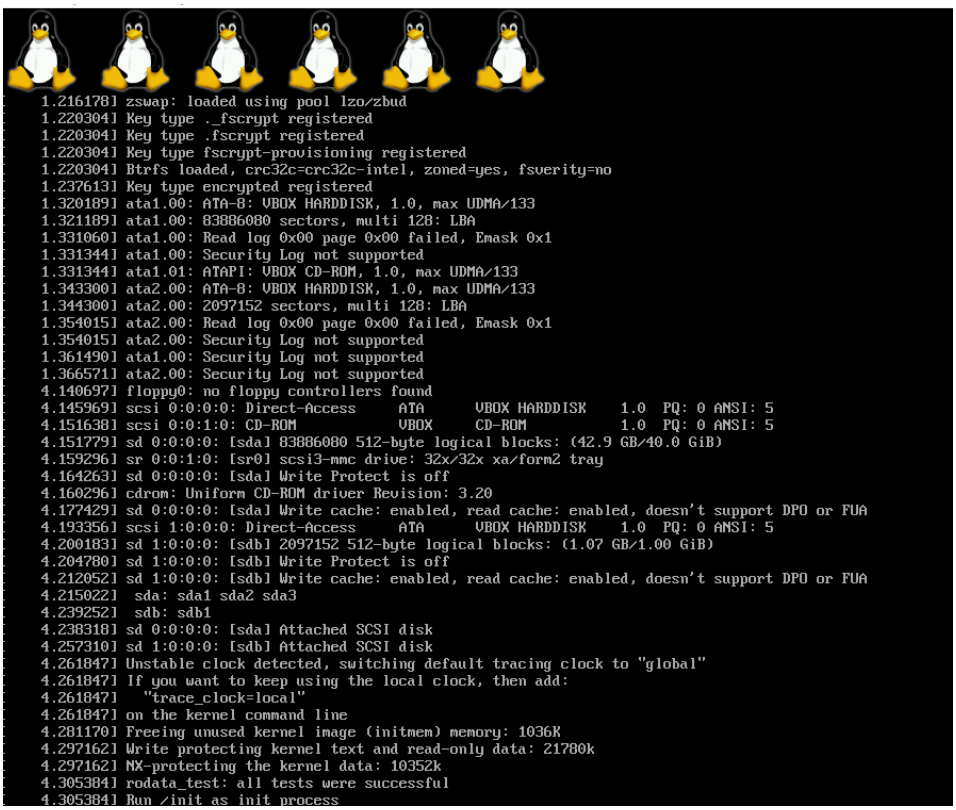


Rysunek 1.20. Wywołanie komendy ***lilo.***

17. Po ponownym uruchomieniu maszyny jest widoczne nasze nowe jądro – ***kernel-old***

Rysunek 1.21. Widok po ponownym uruchomieniu maszyny wirtualnej.

Tutaj napotkałem problem, system nie chciał się uruchomić, zacinał się na instrukcji ***Run /init as init process.***



Rysunek 1.22. Widok zawieszonego systemu.

Próbowałem znaleźć rozwiązanie w internecie jednak nie mogłem dojść do żadnej konkretnej informacji. Postanowiłem zbudować jądro od nowa jednak nic to nie dało – miałem ten sam rezultat. Jako ostatnia próba pobrałem inną wersję jądra – 5.17.14, powtórzyłem powyższe instrukcje i tym razem wszystko zadziałało jak powinno.

Rysunek 1.23. Widok po uruchomieniu systemu z nowo zbudowanym jądrem ***kernel-old2.***

System uruchomił się poprawnie i możemy sprawdzić naszą wersję jądra przy pomocy komendy ***uname -r***

```
Last login: Sun Jun 12 18:24:07 2022 from 192.168.56.1
Linux 5.17.14-smp.
root@localhost:~# uname -r
5.17.14-smp
root@localhost:~#
```

Rysunek 1.24. Widok wersji jądra.

2. Kompilacja przy wykorzystaniu starej metody

1. Przechodzimy do folderu z pobranym jądrem i kopiujemy konfigurację

```
root@localhost:/usr/src/linux-5.17.14# zcat /proc/config.gz > .config
root@localhost:/usr/src/linux-5.17.14# scripts/kconfig/streamline_config.pl > config_strip
using config: '.config'
root@localhost:/usr/src/linux-5.17.14# mv .config config_b
root@localhost:/usr/src/linux-5.17.14# mv config_strip .config
root@localhost:/usr/src/linux-5.17.14#
```

Rysunek 2.1. Kopiowanie konfiguracji.

2. Budowa pliku konfiguracyjnego jądra

```
root@localhost:/usr/src/linux-5.17.14# make oldconfig
  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
```

Rysunek 2.2 Budowa pliku konfiguracyjnego jądra.

3. Jak w przypadku starej metody, jesteśmy zapytani o ustawienie parametrów, ustawiamy wartości domyślne klikając enter.

```
  kmod stress tester (TEST_KMOD) [N/m/?] n
  Test memcat_p() helper function (TEST_MEMCAT_P) [N/m/y/?] n
  Perform selftest on object aggreration manager (TEST_OBJAGG) [N/m/?] n
  Test level of stack variable initialization (TEST_STACKINIT) [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
#
```

Rysunek 2.3. Proces budowy pliku konfiguracyjnego.

4. Budujemy obraz jądra tak samo jak w starej metodzie

```
root@localhost:/usr/src/linux-5.17.14# time make -j6 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
  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/ioctls.h
  WRAP    arch/x86/include/generated/uapi/asm/ioctl.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/resource.h
  WRAP    arch/x86/include/generated/uapi/asm/socket.h
  WRAP    arch/x86/include/generated/uapi/asm/poll.h
  WRAP    arch/x86/include/generated/uapi/asm/sockios.h
  WRAP    arch/x86/include/generated/uapi/asm/termbits.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
  WRAP    arch/x86/include/generated/asm/early_ioremap.h
  WRAP    arch/x86/include/generated/asm/export.h
```

Rysunek 2.4. Proces budowy obrazu jądra.

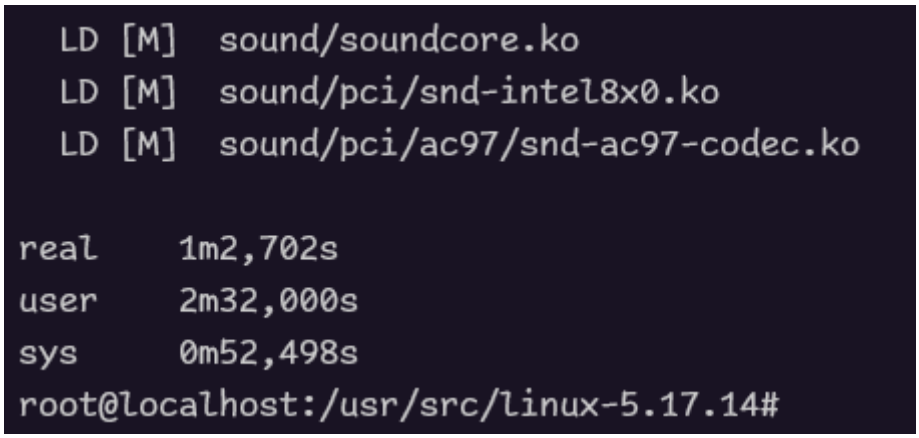
```
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)

real    11m56,981s
user    24m20,987s
sys     7m42,697s
root@localhost:/usr/src/linux-5.17.14#
```

Rysunek 2.5. Zakończenie budowy obrazu jądra.

5. Budujemy moduły jądra tak samo jak w starej metodzie

```
root@localhost:/usr/src/linux-5.17.14# time make -j6 modules
  CALL    scripts/atomic/check-atomics.sh
  CALL    scripts/checksyscalls.sh
  CC [M]  arch/x86/events/amd/power.o
  AS [M]  arch/x86/crypto/serpent-sse2-i586-asm_32.o
  CC [M]  arch/x86/kernel/cpu/mce/inject.o
  CC [M]  arch/x86/crypto/serpent_sse2_glue.o
  CC [M]  arch/x86/platform/iris/iris.o
  CC [M]  arch/x86/events/intel/cstate.o
  CC [M]  kernel/time/test_udelay.o
  AS [M]  arch/x86/crypto/crc32-pclmul_asm.o
  CC [M]  arch/x86/crypto/crc32-pclmul_glue.o
  CC [M]  kernel/trace/ring_buffer_benchmark.o
  CC [M]  arch/x86/platform/scx200/scx200_32.o
```

Rysunek 2.6. budowa modułów jądra.



Rysunek 2.7. Zakończenie budowy modułów jądra.

6. Instalacja modułów

```
root@localhost:/usr/src/linux-5.17.14# make modules_install
  INSTALL /lib/modules/5.17.14-smp/kernel/arch/x86/events/intel/intel-cstate.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/arch/x86/events/rapl.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/acpi/ac.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/acpi/button.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/acpi/video.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/block/loop.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/char/agp/agpgart.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/char/agp/intel-agp.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/char/agp/intel-gtt.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/gpu/drm/drm.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/gpu/drm/drm_kms_helper.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/gpu/drm/drm_ttm_helper.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/gpu/drm/drm_vram_helper.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/gpu/drm/ttm/ttm.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/gpu/drm/vboxvideo/vboxvideo.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/gpu/drm/vmwgfx/vmwgfx.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/i2c/algos/i2c-algo-bit.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/i2c/busses/i2c-piix4.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/i2c/i2c-core.ko
  INSTALL /lib/modules/5.17.14-smp/kernel/drivers/input/evdev.ko
```

Rysunek 2.8. Instalacja modułów za pomocą ***make modules_install.***

7. Sprawdzamy zainstalowane moduły

```
root@localhost:/usr/src/linux-5.17.14# ls /lib/modules/5.17.14-smp/
build@          modules.alias.bin          modules.builtin.bin      modules.dep.bin  modules.softdep      source@
kernel/         modules.builtin            modules.builtin.modinfo  modules.devname  modules.symbols
modules.alias   modules.builtin.alias.bin  modules.dep              modules.order    modules.symbols.bin
root@localhost:/usr/src/linux-5.17.14#
```

Rysunek 2.9. Spis modułów.

8. Kopiujemy pliki jądra do systemu analogicznie do poprzedniej metody.

```
root@localhost:/usr/src/linux-5.17.14# cp arch/x86/boot/bzImage /boot/vmlinuz-new-5.17.14-smp
root@localhost:/usr/src/linux-5.17.14# cp System.map /boot/System.map-new-5.17.14-smp
root@localhost:/usr/src/linux-5.17.14# cp .config /boot/config-new-5.17.14-smp
root@localhost:/usr/src/linux-5.17.14#
```

Rysunek 2.10. Kopiowanie plików jądra.

9. Tworzymy link symboliczny do tablicy symboli

```
root@localhost:/boot# rm System.map
root@localhost:/boot# ln -s System.map-new-5.17.14-smp System.map
root@localhost:/boot#
```

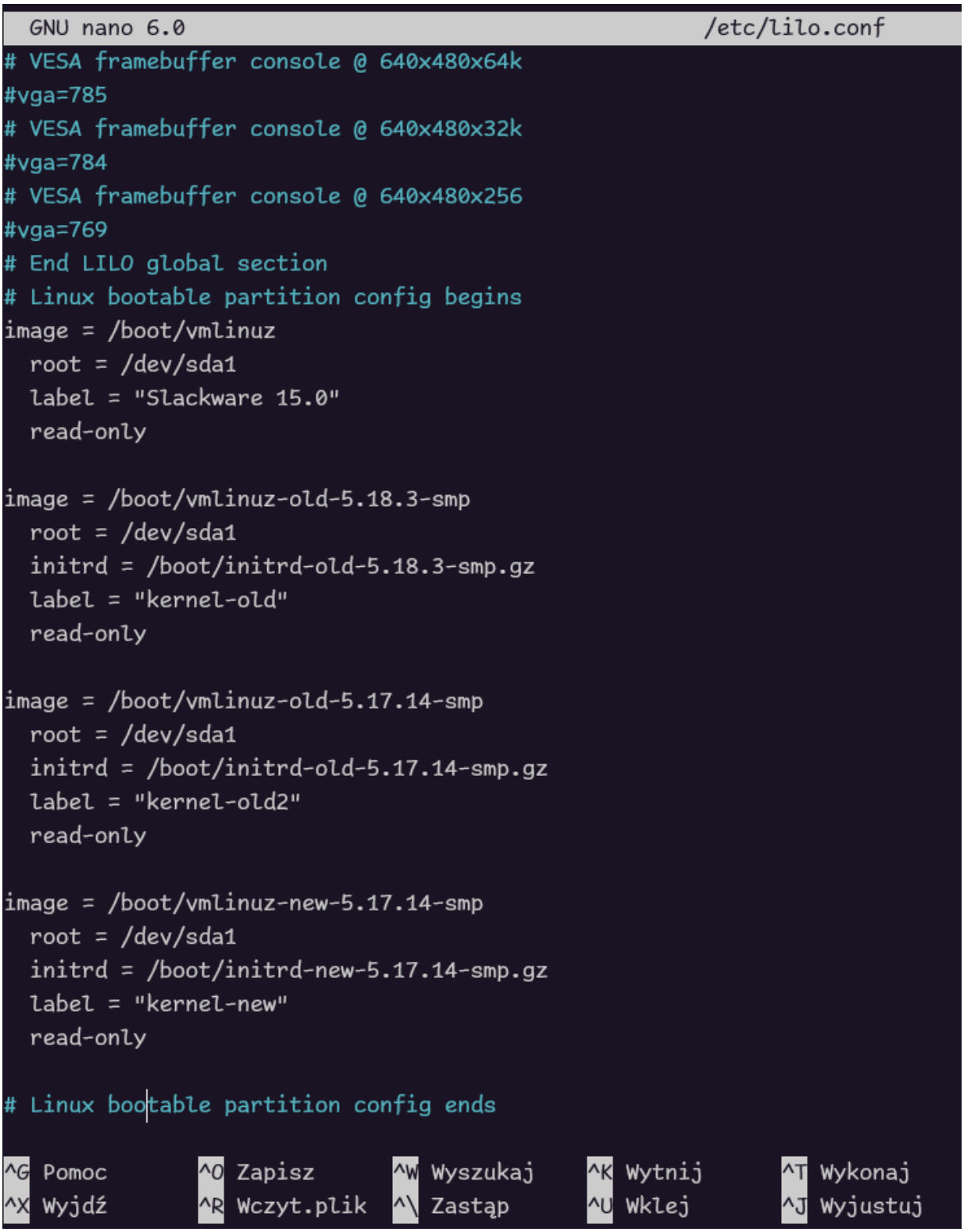
Rysunek 2.11. Tworzenie linku symbolicznego do tablicy symboli.

10. Tworzenie dysku RAM

```
root@localhost:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.17.14-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 5.17.14-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@localhost:/boot# mkinitrd -c -k 5.17.14-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-new-5.17.14-smp.gz
49038 bloków
/boot/initrd-new-5.17.14-smp.gz created.
Be sure to run lilo again if you use it.
root@localhost:/boot#
```

Rysunek 2.12. Tworzenie dysku RAM.

11. Pozostało nam dodać wpis w pliku konfiguracyjnym ***lilo***



Rysunek 2.13. Modyfikacja pliku konfiguracyjnego ***lilo***.

12. Sprawdzamy naszą konfiguracje za pomocą komendy ***lilo***

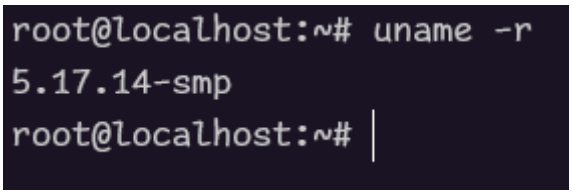
```
root@localhost:/boot# lilo
Warning: LBA32 addressing assumed
Added Slackware_15.0  *
Added kernel-old  +
Added kernel-old2  +
Added kernel-new  +
One warning was issued.
root@localhost:/boot#
```

Rysunek 2.14. Wykonanie komendy ***lilo.***

13. Po ponownym uruchomieniu systemu widzimy opcję ***kernel-new***

Rysunek 2.15. Widok po uruchomieniu systemu.

14. Po wybraniu kernel-new system uruchamia się poprawnie oraz możemy sprawdzić wersję naszego jądra



Rysunek 2.16. Widok wersji jądra

3. Wnioski

Moje odczucia co do przebiegu kompilacji są mieszane. Sam proces nie jest zbytnio skomplikowany, skomplikowane są dopiero błędy które mogą się ewentualnie pojawić, tak jak w moim przypadku nadal nie mam pojęciu czemu nowsza wersja jądra nie była w stanie się uruchomić.

Nowa metoda kompilacji jądra wydaje się o wiele przystępniejsza ponieważ możemy pominąć dużą część konfiguracji, czas kompilacji zarówno modułów (nowa metoda: 1m2s, stara metoda: 1m7s) jak i samego jądra (nowa metoda: 11m56s, stara metoda: 11m11s) były bardzo zbliżone, zatem nie widzę powodu dla którego mielibyśmy wciąż używać starej metody.