Kompilacja jądra Linux - zadanie

Przygotowanie źródeł

Ze strony www.kernel.org pobrano najnowsze stabilne źródła jądra Linux (w wersji **5.18.3**) oraz rozpakowano je w katalogu `/usr/src`.

```
root@localhost:~# cd /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 20:54:13--  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, 2a04:4e42:3::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  19,1MB/s     w 6,5s

2022-06-12 20:54:20 (19,1 MB/s) - zapisano 'linux-5.18.3.tar.xz' [129859840/129859840]

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

```
root@localhost:/usr/src# tar -xvpf linux-5.18.3.tar.xz
```

...

```
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# ls
linux-5.17.8/  linux-5.17.8.tar.xz  linux-5.18.3/  linux-5.18.3.tar.xz
```

Po rozpakowaniu archiwum utworzono dwie kopie folderu `linux-5.18.3`: `linux-5.18.3-n` i `linux-5.18.3-s` (odpowiednio dla nowej i starej metody).

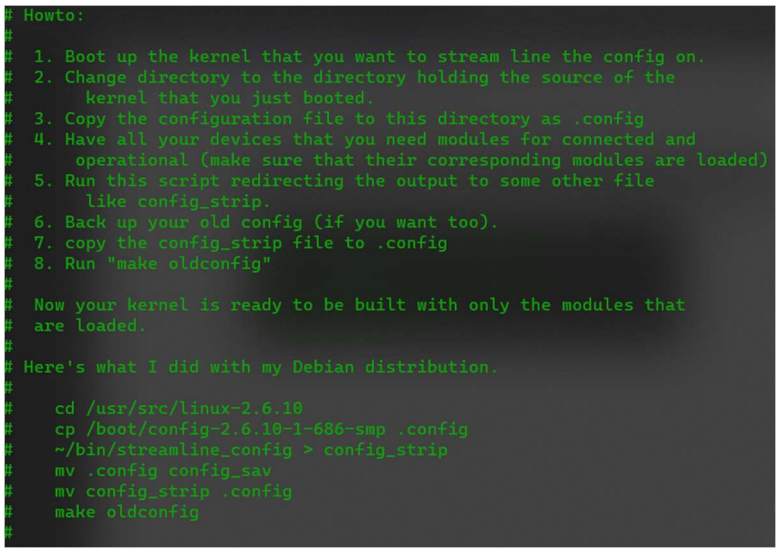
```
root@localhost:/usr/src# cp -r linux-5.18.3/ linux-5.18.3-s/
root@localhost:/usr/src# ls
linux-5.17.8/  linux-5.17.8.tar.xz  linux-5.18.3/  linux-5.18.3-s/  linux-5.18.3.tar.xz
root@localhost:/usr/src# cp -r linux-5.18.3/ linux-5.18.3-n/
root@localhost:/usr/src# ls
linux-5.17.8/  linux-5.17.8.tar.xz  linux-5.18.3/  linux-5.18.3-n/  linux-5.18.3-s/  linux-5.18.3.tar.xz
root@localhost:/usr/src#
```

Jak widać na powyższych zrzutach ekranu, w systemie znajdowały się również źródła w wersji 5.17.8, które były wykorzystane na laboratoriach.

Kompilacja jądra – **nowa** metoda

Po przejściu do odpowiedniego katalogu `linux-5.18.3-n`, skorzystano ze wskazówek zamieszczonych w pliku `scripts/kconfig/streamline_config.pl`.

```
root@localhost:/usr/src# cd linux-5.18.3-n
root@localhost:/usr/src/linux-5.18.3-n# nano scripts/kconfig/streamline_config.pl
```



```
# 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 configuration file to this directory as .config
#  4. Have all your devices that you need modules for connected and
#      operational (make sure that their corresponding modules are loaded)
#  5. Run this script redirecting the output to some other file
#       like config_strip.
#  6. Back up your old config (if you want too).
#  7. copy the config_strip file to .config
#  8. Run "make oldconfig"
#
#  Now your kernel is ready to be built with only the modules that
#  are loaded.
#
# Here's what I did with my Debian distribution.
#
#    cd /usr/src/linux-2.6.10
#    cp /boot/config-2.6.10-1-686-smp .config
#    ~/bin/streamline_config > config_strip
#    mv .config config_sav
#    mv config_strip .config
#    make oldconfig
#
```

Na początku do folderu roboczego skopiowano plik `config` z katalogu `/boot`:

```
root@localhost:/usr/src/linux-5.18.3-n# cp /boot/config .config
root@localhost:/usr/src/linux-5.18.3-n#
```

Następnie wykonano wcześniej wspomniany skrypt:

```
root@localhost:/usr/src/linux-5.18.3-n# ./scripts/kconfig/streamline_config.pl > config_strip
using config: '.config'
root@localhost:/usr/src/linux-5.18.3-n#
```

Potem utworzono kopię zapasową pliku `.config` o nazwie `.config.old`:

```
root@localhost:/usr/src/linux-5.18.3-n# cp .config .config.old
root@localhost:/usr/src/linux-5.18.3-n#
```

Nadpisano plik `.config`:

```
root@localhost:/usr/src/linux-5.18.3-n# mv config_strip .config
root@localhost:/usr/src/linux-5.18.3-n#
```

Bazując na wskazówkach z pliku `scripts/kconfig/streamline_config.pl`, wykonano komendę `make oldconfig`. W przypadku wielu zapytań, wciskano klawisz *Enter*, aby została zastosowana wartość domyślna.

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

...

```
  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-n#
```

Wykonano kompilację jądra używając `bzImage`:

```
root@localhost:/usr/src/linux-5.18.3-n# make -j2 bzImage
```

...

a po ponad godzinie (15:25-16:39) proces zakończył się powodzeniem:

...

```
  LZMA    arch/x86/boot/compressed/vmlinux.bin.lzma
  CC      arch/x86/boot/cpu.o
  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@localhost:/usr/src/linux-5.18.3-n#
```

Następnie przystąpiono do kompilacji modułów:

```
root@localhost:/usr/src/linux-5.18.3-n# make -j2 modules
  CALL    scripts/atomic/check-atomics.sh
  CALL    scripts/checksyscalls.sh
  CC [M]  arch/x86/events/intel/cstate.o
```

która zakończyła się po 8 min (16:46-16:54):

```
  LD [M]  sound/ac97_bus.ko
  LD [M]  sound/core/snd-pcm.ko
  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
root@localhost:/usr/src/linux-5.18.3-n#
```

Kolejnym krokiem było odpowiednie skopiowanie plików do katalogu `/boot`:

- obrazu jądra `bzImage`:

```
root@localhost:/usr/src/linux-5.18.3-n# cp arch/x86/boot/bzImage /boot/vmlinuz-5.18.3-n
```

- tablicy symboli `System.map`:

```
root@localhost:/usr/src/linux-5.18.3-n# cp System.map /boot/System.map-5.18.3-n
```

- nowej konfiguracji zawartej w pliku `.config`:

```
root@localhost:/usr/src/linux-5.18.3-n# cp .config /boot/config-5.18.3-n
```

Dalej zaktualizowano i podlinkowano nową tablicę symboli w katalogu `/boot`:

```
root@localhost:/boot# rm System.map
root@localhost:/boot# ln -s System.map-5.18.3-n System.map
```

Następnie wygenerowano komendę:

```
root@localhost:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.18.3-n
Modules for kernel 5.18.3-n aren't installed.
```

Jak widać wyżej, wyświetlił się monit o tym, że nie ma zainstalowanych modułów dla jądra. W tym celu wywołano komendę `make modules_install`:

```
root@localhost:/boot# cd /usr/src/linux-5.18.3-n
root@localhost:/usr/src/linux-5.18.3-n# make modules_install
  INSTALL /lib/modules/5.18.3-smp/kernel/arch/x86/crypto/crc32-pclmul.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/arch/x86/events/intel/intel-cstate.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/ac.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/battery.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
```

I dalej zadziałało:

```
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#
```

Następnie stworzono *RAMDISK*:

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

Zmodyfikowano plik `lilo.conf`:

```
  GNU nano 6.0                                    /etc/lilo.conf
#vga=769
# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 15.0"
  read-only

image = /boot/vmlinuz-5.18.3-n
  root = /dev/sda1
  label = "Linux nowa metoda"
  initrd = /boot/initrd-5.18.3-n.gz
  read-only
```

Wykonano komendę `lilo`:

```
root@localhost:/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_15.0  *
Fatal: Image name, label, or alias is too long: 'Linux_nowa_metoda'
root@localhost:/boot#
```

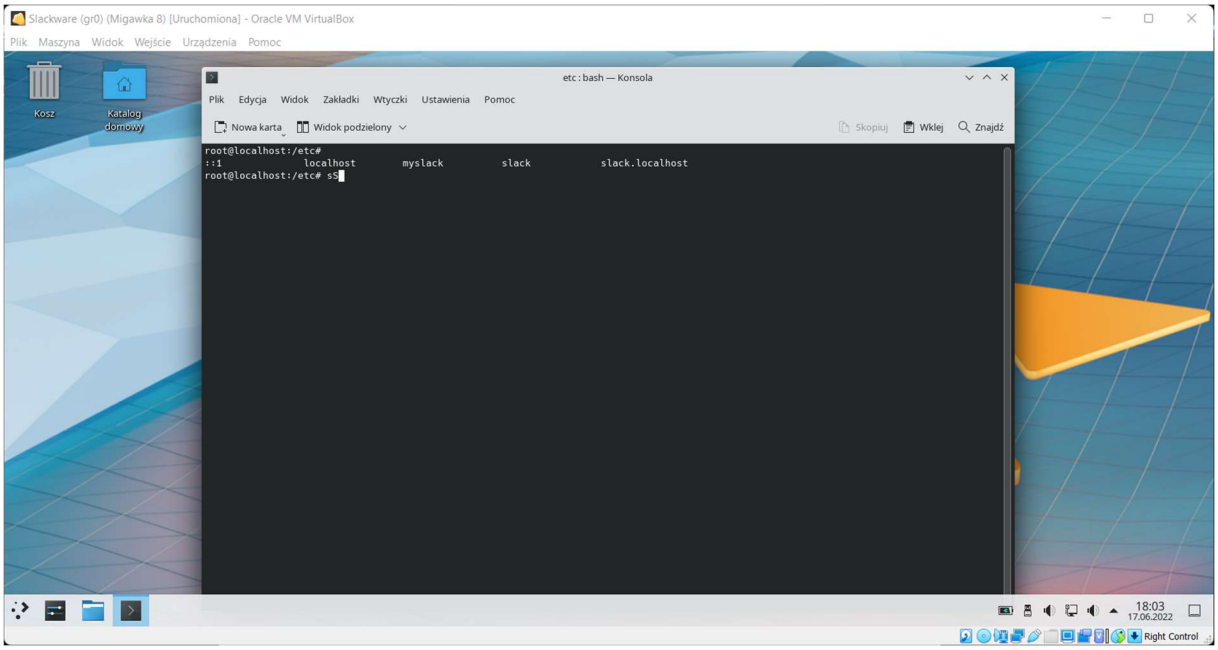
Ukazała się informacja, że pole `label` jest za długie, zatem skrócono je z `Linux_nowa_metoda` na `LinuxN` i wykonano polecenie ponownie:

```
root@localhost:/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_15.0  *
Added LinuxN  +
3 warnings were issued.
root@localhost:/boot#
```

Zrestartowano system.

Po restarcie ukazał się ekran bootowania, gdzie znalazła się pozycja `LinuxN`:

Po wybraniu `LinuxN` uruchamianie systemu zakończyło się sukcesem (jak widać niżej). Co ciekawe, gdy system był uruchamiany domyślnie z pozycji `Slackware_15.0` (tej z zajęć), to interfejs graficzny nie ładował się odpowiednio i trzeba było wcisnąć kombinację *Ctrl + Alt + F1*, zalogować się i ręcznie uruchomić tryb graficzny. W przypadku `LinuxN` interfejs graficzny ładuje się sam i nic nie trzeba robić. Było to miłe zaskoczenie po kilku godzinach pracy nad pierwszą częścią zadania.



Kompilacja jądra – **stara** metoda

Po przejściu do katalogu `linux-5.18.3-s` skopiowano starą konfigurację jądra do katalogu roboczego:

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

W kolejnym kroku skorzystano z komendy `make localmodconfig`, aby wygenerować plik konfiguracyjny. Tym razem nie wystąpiły żadne zapytania.

```
root@localhost:/usr/src/linux-5.18.3-s# 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'
#
# configuration written to .config
#
root@localhost:/usr/src/linux-5.18.3-s#
```

Następnie wykonano większość podobnych czynności, jak w nowej metodzie. Wszystkie kolejne kroki przedstawione są poniżej na zrzutach ekranu:

```
root@localhost:/usr/src/linux-5.18.3-s# make -j2 bzImage
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_32.h
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_64.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
```

...

po około pół godziny (21:55-22:26) proces kompilacji jądra przebiegł pomyślnie:

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

Kompilacja modułów:

```
root@localhost:/usr/src/linux-5.18.3-s# make -j2 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/intel-cstate.o
```

... po upływie tylko 3 min (22:30-22:33) moduły zostały zbudowane:

```
  LD [M]  sound/ac97_bus.ko
  LD [M]  sound/core/snd-pcm.ko
  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
root@localhost:/usr/src/linux-5.18.3-s#
```

Wykonanie (w razie czego) `make modules_install`:

```
root@localhost:/usr/src/linux-5.18.3-s# make modules_install
  INSTALL /lib/modules/5.18.3-smp/kernel/arch/x86/crypto/crc32-pclmul.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/arch/x86/events/intel/intel-cstate.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/ac.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/battery.ko
  INSTALL /lib/modules/5.18.3-smp/kernel/drivers/acpi/button.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@localhost:/usr/src/linux-5.18.3-s#
```

Skopiowano odpowiednie pliki do katalogu `/boot`:

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

Zaktualizowano tablicę symboli:

```
root@localhost:/boot# rm System.map
root@localhost:/boot# ln -s System.map-5.18.3-s System.map
```

Skonfigurowanie *RAMDISK*:

```
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#
```

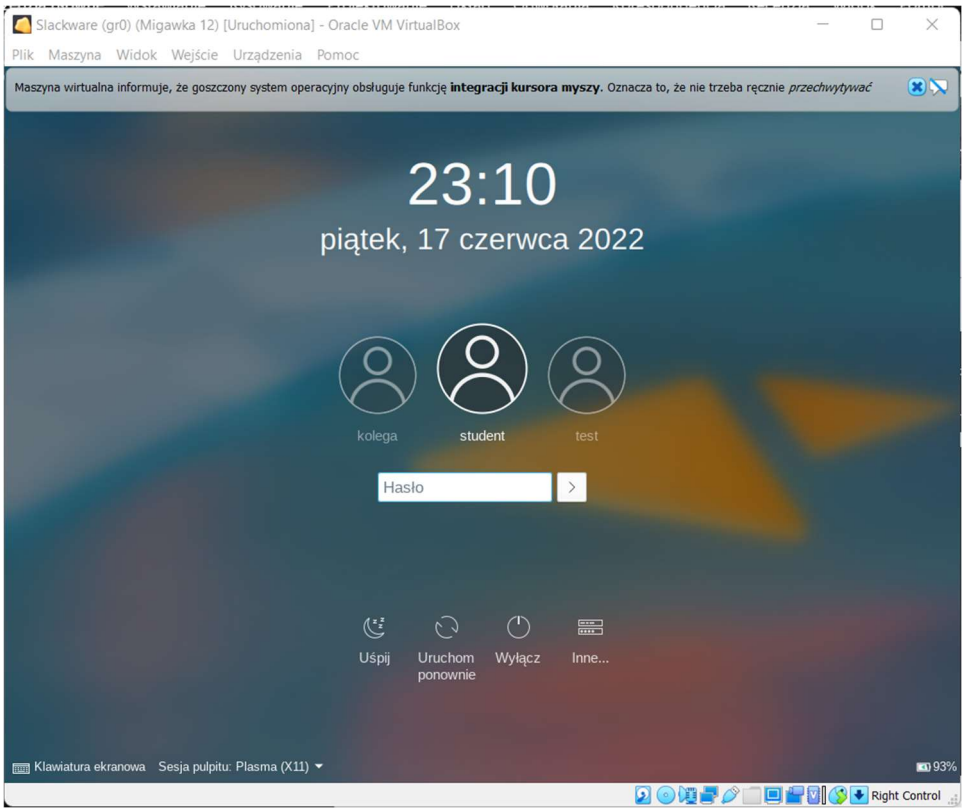
```
root@localhost:/boot# mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-5.18.3-s.gz
49039 bloków
/boot/initrd-5.18.3-s.gz created.
Be sure to run lilo again if you use it.
root@localhost:/boot#
```

Zmodyfikowanie pliku konfiguracyjnego `lilo.conf`:

```
  GNU nano 6.0                                          /etc/lilo.conf
# VESA framebuffer console @ 640x480x256
#vga=769
# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 15.0"
  read-only

image = /boot/vmlinuz-5.18.3-n
  root = /dev/sda1
  label = "LinuxN"
  initrd = /boot/initrd-5.18.3-n.gz
  read-only

image = /boot/vmlinuz-5.18.3-s
  root = /dev/sda1
  label = "LinuxS"
  initrd = /boot/initrd-5.18.3-s.gz
  read-only

# Linux bootable partition config ends
```

Zapisano powyższy plik i zrestartowano system. Po restarcie, na liście nie pojawił się `LinuxS`, ale to dlatego, że zapomniano wywołać komendy `lilo`. Po zalogowaniu się i wywołaniu tej komendy (w systemowym terminalu, żeby już ponownie nie logować się przez *SSH*) ponownie uruchomiono maszynę.

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



Jak widać na powyższych rysunkach, wszystko przebiegło pomyślnie.

Podsumowanie

Katalogi robocze, zarówno dla nowej i starej metody, zostały umieszczone w serwisie GitHub (https://github.com/Matek0611/LinuxZadanieKernel). Zostały one jednak podzielone na aż cztery części, gdyż platforma ta akceptuje pliki do max. 100 MB, a niestety sumaryczna wielkość całego zarchiwizowanego katalogu miała wielkość ok. 370 MB. Chciałem zamiast tego wrzucić te foldery bez spakowania, ale występował jeszcze jakiś inny błąd, więc pozostałem przy omawianym rozwiązaniu.

Po wykonaniu kompilacji jądra systemu Linux dwoma metodami, mam kilka przemyśleń. Znaczącą różnicą w procesie kompilacji był czas jej przebiegu. Dla nowej metody tworzenie obrazu `bzImage` zajęło aż ponad godzinę, gdy dla starej – tylko połowę tego czasu. Wykonanie instalacji modułów, w przypadku starej metody, okazało się 2,6 razy szybsze niż dla tej drugiej. Z drugiej strony, zaletą nowej wersji niewątpliwie są wskazówki zawarte w pliku `streamline_config.pl`, dzięki którym można dokładniej i szybciej przejść początkowy etap całego procesu. W pozostałych przypadkach nie zauważyłem zbytnich różnic. Według mnie obie metody są dobre, jednakże na swoim sprzęcie skorzystałbym raczej ze starej, gdyż rezultaty następują prędzej.