Architektura komputerów 2 – projekt

Linux keyboard driver

1. Wprowadzenie

Podczas projektu będziemy pracować na otwartym oprogramowaniu Linux – które daje możliwość poznania działania systemu operacyjnego od środka i dogłębnej analizy oraz modyfikacji jądra dostarczającego zasoby dla innych procesów w systemie .

Główną częścią systemu operacyjnego Linux jest jego jądro – Linux kernel, którego zadaniami są:

- zarządzanie pamięcią operacyjną (przydzielanie, zwalnianie oraz ochrona pamięci)

- zarządzanie procesami (tworzenie, niszczenie procesów, komunikacja miedzyprocesorowa)

- **obsługa sterowników urządzeń (ładowanie i usuwanie sterowników)**

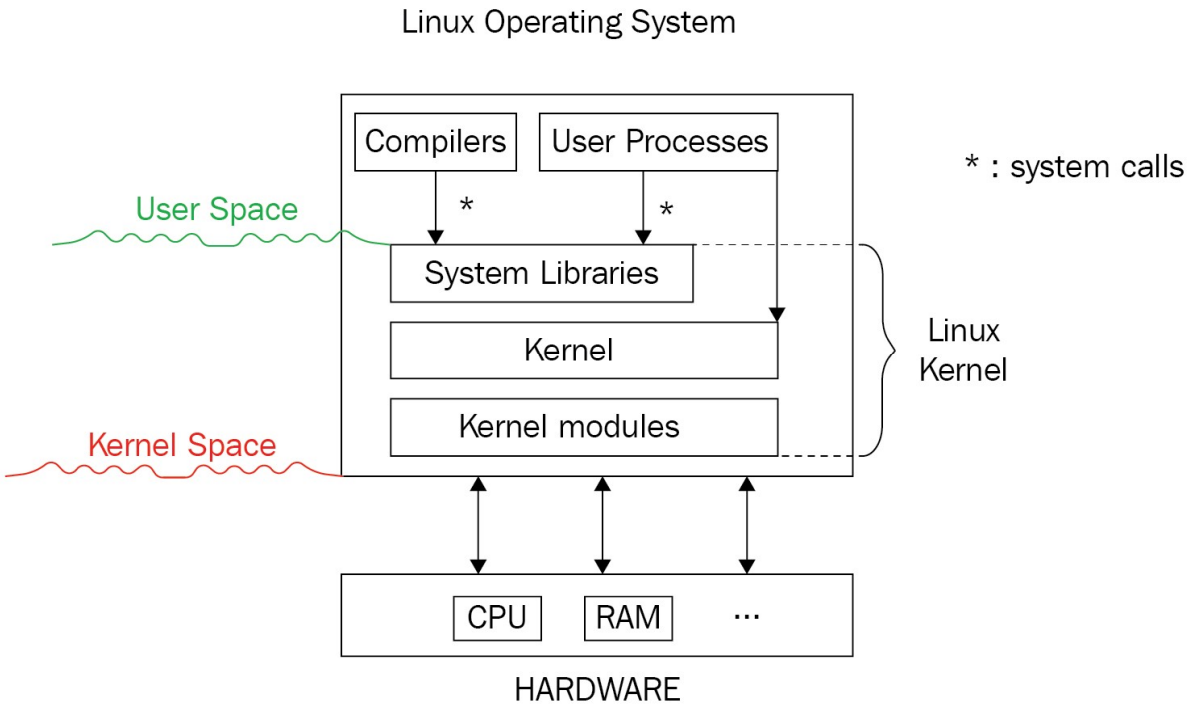
- obsługa wywołań systemowych

Kod wywoływany przez system na procesorze może zostać uruchomiony w dwóch trybach:

- Kernel mode
- User mode

Procesy wywoływane z poziomu użytkowników mają ograniczony dostęp do poszczególnych elementów hardware (CPU, pamięć), przez co szkody wyrządzone z tego poziomu nie będą tak szkodliwe jak wywołane przez w części kernela, gdzie nie ma żadnych restrykcji co do możliwości wykonywanych procesów.

1.1 Przedstawienie warstw systemu operacyjnego Linux



Większa część jądra została napisana w języku C (niektóre wstawki w assemblerze), dlatego podczas przygotowywania sterownika do obsługi klawiatury również będziemy posługiwać się językiem C.

Aby rozszerzyć podstawowe funkcje jądra można dodać do niego moduły. Dużą zaletą tej funkcjonalności jest możliwość załadowania oraz usunięcia tej części kodu, bez potrzeby restartowania całego systemu. Jednym z nich będzie nasz sterownik linux keyboard driver, który umożliwi nam przechwytywanie wciśnięć klawiatury.

Każdy moduł kernela potrzebuje posiadać co najmniej dwie funkcje:

- Funkcja załadowania modułu `init_module()`
- Funkcja usunięcia modułu `cleanup_module()`

Dodatkowo do modułu należy załadować następujące biblioteki:

- `linux/module.h`

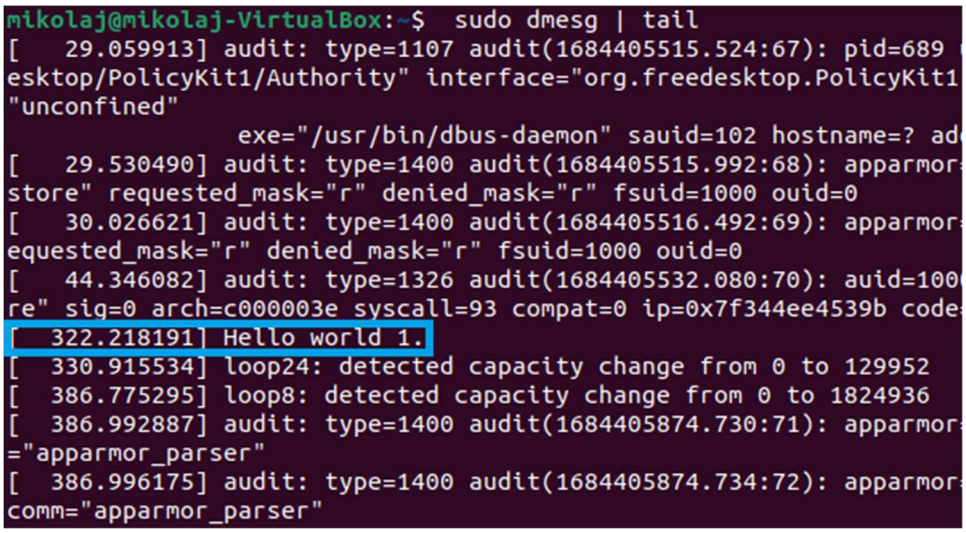
Zapoznanie z obsługą sterowników i pracą w kernel space.

Pierwszym krokiem w celu zgłębienia tematu będzie napisanie najprostszego modułu „Hello, World”, który po załadowaniu w logach kernela wypisuje za pomocą funkcji `printk()` „Hello, World” a przy usunięciu modułu wypisuje „Goodbye, World” .

Chcąc załadować moduł, należy napisać plik Makefile, a następnie zbudować go komendą `make`. Powstały w ten sposób plik z rozszerzeniem .ko należy załadować do kernela komendą `insmod`. Za pomocą funkcji `lsmod` możemy zobaczyć załadowane moduły.

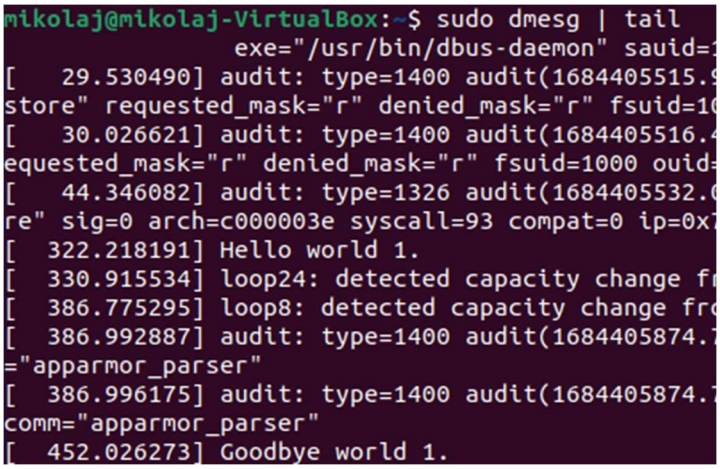
```
mikolaj@mikolaj-VirtualBox:~$ sudo insmod ./hello-1.ko
[sudo] password for mikolaj:
mikolaj@mikolaj-VirtualBox:~$ lsmod
Module                  Size  Used by
hello_1                16384  0
isofs                  53248  1
vboxnetadp             28672  0
vboxnetflt             28672  0
vboxdrv               573440  2 vboxnetadp,vboxnetflt
intel_rapl_msr         20480  0
binfmt_misc            24576  1
snd_intel8x0           49152  2
snd_ac97_codec        176128  1 snd_intel8x0
ac97_bus               16384  1 snd_ac97_codec
intel_rapl_common      40960  1 intel_rapl_msr
snd_pcm               155648  2 snd_intel8x0,snd_ac97_codec
crct10dif_pclmul       16384  1
snd_seq_midi           20480  0
ghash_clmulni_intel    16384  0
nls_iso8859_1          16384  1
joydev                 32768  0
snd_seq_midi_event     16384  1 snd_seq_midi
aesni_intel           376832  0
crypto_simd            16384  1 aesni_intel
cryptd                 24576  2 crypto_simd,ghash_clmulni_intel
snd_rawmidi            45056  1 snd_seq_midi
snd_seq                77824  2 snd_seq_midi,snd_seq_midi_event
snd_seq_device         16384  3 snd_seq,snd_seq_midi,snd_rawmidi
input_leds             16384  0
snd_timer              40960  2 snd_seq,snd_pcm
snd                   114688  11 snd_seq,snd_seq_device,snd_intel8x0,snd_timer,snd_ac97_codec,snd_pcm,snd_rawmidi
serio_raw              20480  0
soundcore              16384  1 snd
vboxguest              45056  0
mac_hid                16384  0
sch_fq_codel           24576  2
vmwgfx                372736  2
```

Dmesg | tail -> pozwala na zobaczenie kilku ostatnich logów kernela



Na powyższym zrzucie ekranu widać, że moduł po załadowaniu zgodnie z oczekiwaniami wypisał w logach kernela „Hello world 1”.

W celu usunięcia modułu należy użyć komendy `rmmod`.



Po ponownym sprawdzeniu logów, możemy zauważyć, informacje o wyładowaniu modułu „Goodbye world 1”.

Kolejnym etapem w realizacji naszego projektu było napisanie character device driver.

Każdemu sterownikowi przypisany jest unikalny numer Major . Wszystkie urządzenie z tym samym numerem Major, kontrolowane są przez ten sam sterownik. Numer Minor mówi sterownikowi jakiego rodzaju jest urządzenie, które kontroluje.

Wpierw załadujemy skompilowany moduł do kernela.

```
antek@antek-VirtualBox:~$ sudo insmod dev_nr.ko
```

Wyświetlając logi w kernelu, możemy zobaczyć, że urządzenie zostało zarejestrowane.

```
[ 5598.501836] Inicjalizacja modulu
[ 5598.501841] dev_nr - zarejsetrowano urzadzenie numer Major: 90, Minor: 0
```

Za pomocą polecenia `mknod` tworzymy urządzenie o podanych numerach.

```
antek@antek-VirtualBox:~$ sudo mknod /dev/my_device c 90 0
```

Następnie przy użyciu `ls -al` wyświetlamy pliki znajdujące się w podanej ścieżce.

```
antek@antek-VirtualBox:~$ ls /dev/my_device -al
crw-r--r-- 1 root root 90, 0 maj 28 16:20 /dev/my_device
```

Odpalamy program test, który otwiera i zamyka nasze urządzenie.

```
antek@antek-VirtualBox:~$ ./test
Opening was successfull!
antek@antek-VirtualBox:~$ sudo dmesg | tail -n 2
[ 6243.561187] dev_nr - wywolano otwarcie!
[ 6243.561212] dev_nr - wywolano zamkniecie!
```

Chcąc sprawdzić czy nasze nowo dodane urządzenie znajduje się na liście urządzeń, sprawdzamy zawartość pliku devices (jak widać nasze urządzenie my_dev_nr jest obecne na liście).

```
antek@antek-VirtualBox:~$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  5 ttyprintk
  6 lp
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 89 i2c
 90 my_dev_nr
 99 ppdev
```

Po usunięci modułu możemy zauważyć ze nasze urządzenie zniknęło z listy.

(Ostatecznie zostaliśmy uświadomieni, że dalsza implementacja character device driver nie jest celem naszego projektu i musieliśmy zrobić krok wstecz).

Natrafiliśmy na pomysł napisania sterownika do klawiatury, wzorując się na linuxowym sterowniku autorstwa Vojtecha Pavlika (https://github.com/torvalds/linux/blob/master/drivers/hid/usbhid/usbkbd.c), modyfikując go z dodatkową funkcją rozpoznawania sekwencji konami. Jednakże napotkaliśmy na problem - w jaki sposób wyładować ten moduł i zastąpić go naszym? W celu rozwiązania problemu, skonsultowaliśmy się z prowadzącym, który naprowadził nas na lepsze rozwiązanie. Naszym planem było napisanie dodatkowego modułu, który będzie współgrał ze wbudowanym sterownikiem klawiatury. Jego zadaniem zakodowanie poszczególnych przycisków, rozpoznanie sekwencji (góra, góra, dół, dół, lewo, prawo, lewo, prawo), obsługa resetu sekwencji przy zbyt długim czasie oczekiwania na kolejny klawisz lub przy nieprawidłowym układzie wprowadzonych znaków.

Postanowiliśmy działać na szkielecie poprzedniego modułu kernela, który wypisywał „Hello world 1" w logach podczas inicjalizacji oraz „Goodbye World 1" podczas wyładowania modułu z jądra.

Pierwszym naszym działającym programem był `konami_v1`, w którym udało nam się wykrywać zadaną sekwencję klawiszy przez użytkownika i wypisywać tą informację w logach kernela. W tym programie zadbaliśmy również o delay, dzięki któremu jeśli użytkownik zbyt długo czekał (3000 ms) z wciśnięciem klawisza z sekwencji, układ resetował się.

W drugim prototypie naszego sterownika (`konami_v2`) udało nam się zadbać również o resety sekwencji podczas błędnie wprowadzonej kolejności przycisków.

Ostatnim potrzebnym korkiem do osiągniecia, było dodatnie jakiegoś easter egga, który odpalałby się po poprawnie wprowadzonej sekwencji. Pierwszym pomysłem było użycie brzęczyka, który zacząłby wydawać dźwięki. Jednakże zrezygnowaliśmy z tego pomysłu ze względu na pracę na maszynie wirtualnej, która mogłaby sprawiać problemy z obsługą owego urządzenia. W celu utworzenia jakiegokolwiek easter egga, należało stworzyć dodatkowy wątek kernela, który wykonywałby żądane działania.

Pierwszą ”działającą” koncepcją było uruchomienie zewnętrznego programu (`konami_v3`). Program ten został napisany w języku C, jego zadaniem było otwarcie pliku oraz zapisanie do niego kilku informacji. Na tym poziomie natrafiliśmy na problem, który polegał tworzeniu nowego wątku w nieprawidłowym miejscu co skutkowało działaniem programu tylko w niektórych przypadkach. Innym problemem było prawidłowe wyładowanie modułu z jądra (po wyładowaniu moduł nadal pojawiał się na liście modułów).

W ulepszonej wersji (`konami_v4`), udało nam się zasymulować wciśniecie przycisku klawiatury przy pomocy funkcji `input_report_key`. W zewnętrznym wątku symulujemy w nieskończoność wciśniecie przycisku `'a'`. Prawidłowo tworzymy wątek w funkcji inicjalizującej moduł, a po wykryciu konami jedynie go budzimy, jednakże moduł był napisany w sposób złośliwy – mianowicie nie dało się zatrzymać symulacji wciskania przycisku `'a'` – w celu resetu potrzebne było ponowne uruchomienie maszyny wirtualnej.

Finalną wersją naszego sterownika został (`konami_v5_final`), w którym udało się naprawić poprzednie błędy oraz dodać kilka nowych ulepszeń.

Ostatecznym easter eggiem został taki o kot:

Kot ten, po wykryciu sekwencji, wypisywany jest w nieskończoność.

Dodatkową funkcjonalnością jest możliwość zatrzymania wypisywania kota. Można to osiągnąć poprzez wciśniecie klawisza `'s'` na klawiaturze.

Podsumowując, z radością stwierdzamy, że udało nam się osiągnąć początkowe cele projektu – stworzenie sterownika do klawiatury, który będzie zdolny do rozpoznania sekwencji Konami Code oraz wywołania easter egg. Jednakże, podczas trwania prac napotkaliśmy na wiele wyzwań zarówno na etapie koncepcji działania samego sterownika, jak i podczas implementacji. Uważamy jednak, że te trudności to nieodłączna część tego typu projektów. Sam projekt pozwolił nam zgłębić techniczną wiedze na temat – jądra systemu operacyjnego linux, sposobu działania sterowników oraz projektowania interaktywnych funkcji (nasz easter egg), ale również sprawdzić naszą zdolność do pracy w zespole (efektywna komunikacja, podział obowiązków, koordynacja i zarządzanie czasem).

Źródła:

- https://subscription.packtpub.com
- https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel
- https://tldp.org/LDP/lkmpg/2.6/html/lkmpg.html#AEN27
- https://github.com/Johannes4Linux/Linux_Driver_Tutorial