Prosty sterownik

Michał “mina86” Nazarewicz

http://mina86.com/
mina86@mina86.com

6 marca 2010

O czym będzie prezentacja?

- Szkielet sterownika.
- Plik urządzenia.
- Operacje `read(2)` i `write(2)`.
- Operacja `lseek(2)`.
- Zarządzanie pamięcią.
- Wzajemne wykluczanie.
- Parametry modułu.

Dla kogo jest prezentacja?

- Dla chętnego napisać sterownik dla Linuksa...
- ...lub ciekawego jak to wygląda od środka.
- Wskazana znajomość języka C...
- ...lub innego języka strukturalnego...
- ...w razie wątpliwości **zachęcam do zadawania pytań**.

Kim jestem?

Czyli jak ja śmiem mówić o sterownikach.

- Student PW.
- Od 8 lat fan Slackware'a.
- Od ponad roku pracownik grupy platformowej SPRC.
- Kilka patchy w jądrze.
- Zatem „coś" wiem o sterownikach...
- ...i postaram się zagadnienie przybliżyć.

Hello World

- Prosty sterownik.
- W zasadzie to sam szkielet.
- Przy załądowaniu i usunięciu wypisuje tekst.
- Poza tym nie robi nic. :)

Hello World, deklaracje

- Początkowe deklaracje:

`hello.c` (1–10)

```
1  /*
2   * Hello world driver.
3   * Copyright (c) 2010 Michal Nazarewicz (mina86@mina86.com)
4   */

6  #include <linux/module.h>

8  MODULE_DESCRIPTION("Simple hello world driver");
9  MODULE_AUTHOR("Michal Nazarewicz");
10 MODULE_LICENSE("GPL");
```

Hello World, wejście

- Funkcja inicjująca:

`hello.c` (13–18)

```
13 static int __init hello_init (void)
14 {
15         printk(KERN_INFO "hello, world\n");
16         return 0;
17 }
18 module_init( hello_init )
```

Hello World, wyjście

- Funkcja czyszcząca:

`hello.c` (20–99)

```
20 static void __exit hello_exit (void)
21 {
22     printk(KERN_INFO "Goodbye, cruel world, I'm leaving you today\n");
23     printk(KERN_INFO "Goodbye, goodbye, goodbye\n");
24     printk(KERN_INFO "Goodbye, all you people\n");
25     printk(KERN_INFO "There's nothing you can say\n");
26     printk(KERN_INFO "To make me change my mind, goodbye\n");
27 }
28 module_exit( hello_exit )
```

Hello World, budowanie

- Plik `Makefile`:
  - `obj-m += hello.o`
- Komenda budująca:
  - `make -C /usr/src/linux "M=$PWD" modules`

Hello World, testowanie

- Wczytywanie modułu:
  - `insmod hello.ko`
- Usuwanie modułu:
  - `rmmod hello`
- Sprawdzanie komunikatów:
  - `dmesg`
  - `dmesg -c`

Zero

- Proste urządzenie znakowe.
- Imituje standardowe urządzenie `/dev/zero`.
- Zapisywane dane są ignorowane.
- Przy odczycie same zera.

Zero, deklaracje

- Dołączone nowe nagłówki:

zero.c (1–13)

```
1  /*
2   * Zero driver
3   * Copyright (c) 2010 Michal Nazarewicz (mina86@mina86.com)
4   */

6  #include <linux/module.h>
7  #include <linux/miscdevice.h>
8  #include <linux/fs.h>
9  #include <linux/uaccess.h>

11 MODULE_DESCRIPTION("Example driver doing what /dev/zero does");
12 MODULE_AUTHOR("Michal Nazarewicz");
13 MODULE_LICENSE("GPL");
```

Zero, init i exit

- Funkcja inicjująca i czyszcząca:

zero.c (53–99)

```
53 static int __init zero_init (void)
54 {
55     int ret = misc_register (&zero_dev);
56     printk (KERN_INFO "zero: loading, misc_register returned %d\n", ret);
57     return ret;
58 }
59 module_init( zero_init )

61 static void __exit zero_exit (void)
62 {
63     int ret = misc_deregister (&zero_dev);
64     printk (KERN_INFO "zero: unloading, misc_deregister returned %d\n",
           ret);
65 }
66 module_exit( zero_exit )
```

Zero, struktury/deskryptory

- Opis urządzenia:

zero.c (46–50)

```
46 static struct miscdevice zero_dev = {
47     .minor = MISC_DYNAMIC_MINOR,
48     .name = "myzero",
49     .fops  = &zero_fops,
50 };
```

- Spis operacji plikowych:

zero.c (40–44)

```
40 static const struct file_operations zero_fops = {
41     .read    = zero_read,
42     .write   = zero_write,
43 };
```

Zero, operacja `write(2)`

- Operacja `write(2)` zwyczajnie ignoruje dane wejściowe:

`zero.c` (31–37)

```
31 static ssize_t zero_write(struct file *file, const char __user *buf,
32                           size_t len, loff_t *offp)
33 {
34     printk(KERN_INFO "zero: write(%zu)\n", len);
35
36     return len;
37 }
```

Zero, operacja `read(2)`

- Operacja `read(2)` wypełnia bufor użytkownika zerami:

`zero.c` (16–29)

```
16 static const char zeros[PAGE_SIZE];

18 static ssize_t zero_read(struct file *file, char __user *buf,
19                           size_t len, loff_t *offp)
20 {
21     ssize_t ret = min(len, sizeof zeros);
22     printk(KERN_INFO "zero: read(%zu (%zd))\n", len, ret);

24     if (likely(ret))
25         if (unlikely(__copy_to_user(buf, zeros, ret)))
26             ret = -EFAULT;

28     return ret;
29 }
```

Zero, testowanie

- Budowanie, wczytywanie i usuwanie jak w „Hello World”.
- Zapis:
  - `echo foo >/dev/myzero; dmesg | tail -n1`
- Odczyt:
  - `head -c1024 /dev/myzero | hexdump -C; dmesg | tail -n1`
  - `head -c8000 /dev/myzero | hexdump -C; dmesg | tail -n2`

Memory

- Urządzenie znakowe pozwalające na zapamiętywanie danych.
- Zapisywane dane są umieszczane w buforze.
- Dane z bufora można następnie odczytać.
- Dane są wspólne dla każdego otworzonego pliku.
- Dane nie są gubione, gdy wszystkie pliki są zamykane.

- Nagłówki, a także funkcja inicjująca i czyszcząca są identyczne jak w porzednim module.

Memory, definicje

- Modyfikowalny bufor zamiast zer:

mem-basic.c (16–17)

```
16 static char memory[PAGE_SIZE];
```

- Nowa operacja plikowa, `llseek`:

mem-basic.c (72–76)

```
72 static const struct file_operations mem_fops = {
73     .llseek  = mem_llseek,
74     .read    = mem_read,
75     .write   = mem_write,
76 };
```

Memory, operacja `read(2)`

- Operacja `read(2)` wczytuje dane z bufora:

`mem-basic.c` (35–49)

```
35 static ssize_t mem_read(struct file * file, char __user *buf,
36                          size_t len, loff_t *offp)
37 {
38     printk(KERN_INFO "mem: read(%zu)\n", len);
39
40     if (*offp >= sizeof memory || !len)
41         return 0;
42
43     len = min(len, (size_t)(sizeof memory - *offp));
44     if (unlikely(__copy_to_user(buf, memory + *offp, len)))
45         return -EFAULT;
46
47     *offp += len;
48     return len;
49 }
```

Memory, operacja `write(2)`

- Operacja `write(2)` zapisuje dane do bufora:

`mem-basic.c` (52–69)

```
52 static ssize_t mem_write(struct file *file, const char __user *buf,
53                          size_t len, loff_t *offp)
54 {
55     printk(KERN_INFO "mem: write(%zu)\n", len);
57     if (!len)
58         return 0;
60     if (*offp >= sizeof memory)
61         return -ENOSPC;
63     len = min(len, (size_t)(sizeof memory - *offp));
64     if (unlikely(__copy_from_user(memory + *offp, buf, len)))
65         return -EFAULT;
67     *offp += len;
68     return len;
69 }
```

Memory, operacja `lseek(2)`

- Wspomniana operacja `llseek` implementująca `lseek(2)`:

`mem-basic.c` (19–32)

```
19 static loff_t mem_llseek(struct file * file, loff_t off, int whence)
20 {
21     switch (whence) {
22     case SEEK_CUR:
23         off += file->f_pos;
24         break;
25     case SEEK_END:
26         off += sizeof memory;
27     }
28     if (off < 0)
29         return -EINVAL;
30     file->f_pos = off;
31     return off;
32 }
```

Memory, testowanie

- `echo Lorem ipsum dolor sit amet >/dev/mymem`
- `head -n1 /dev/mymem`
- `cat /dev/urandom >/dev/mymem`
- `hexdump -C /dev/mymem`

Growable

- Bufor na początku pusty.
- Bufor rośnie, gdy wpisuje się dane.
- Określony limit rozmiaru (1 MiB).

Growable, deklaracje

- Dodany nowy nagłówek:

mem-grow.c (1–14)

```
1  /*
2   * "Memory" driver, growable
3   * Copyright (c) 2010 Michal Nazarewicz (mina86@mina86.com)
4   */

6  #include <linux/module.h>
7  #include <linux/miscdevice.h>
8  #include <linux/fs.h>
9  #include <linux/uaccess.h>
10 #include <linux/vmalloc.h>

12 MODULE_DESCRIPTION("Example growable 'memory' driver");
13 MODULE_AUTHOR("Michal Nazarewicz");
14 MODULE_LICENSE("GPL");
```

Growable, podstawy zarządzania pamięcią

- Pamięć alokowana dynamicznie zamiast statycznego bufora:

mem-grow.c (17–20)

```
17 #define mem_max_size (size_t)(1 << 20)
18 static  size_t  mem_capacity;
19 static  size_t  mem_size;
20 static  char *memory;
```

- Konieczność zwalniania przy wyjściu:

mem-grow.c (127–133)

```
127 static  void  __exit  mem_exit(void)
128 {
129         misc_deregister (&mem_dev);
130         vfree (memory);
131 }
132 module_exit(mem_exit)
```

Growable, operacja `write(2)`

- Operacja `write(2)` alokuje pamięć przy zapisie:

`mem-grow.c` (87–104)

```
87      if (!len)
88          return 0;

90      if (*offp >= mem_max_size)
91          return -ENOSPC;

93      ret = ensure_capacity(min((size_t)(*offp+len), mem_max_size));
94      if (unlikely(ret < 0))
95          return ret;

97      len = min(len, (size_t)(mem_capacity - *offp));
98      if (unlikely(__copy_from_user(memory + *offp, buf, len)))
99          return -EFAULT;

101     *offp += len;
102     mem_size = max((size_t)*offp, mem_size);
103     printk(KERN_INFO "mem: new size %zu\n", mem_size);
104     return len;
```

Growable, alokacja pamięci

- Co robi `ensure_capacity()`?

mem-grow.c (23–44)

```
23 static int ensure_capacity( size_t capacity )
24 {
25     capacity = (capacity + ~PAGE_MASK) & PAGE_MASK;

27     if (mem_capacity < capacity) {

   ⋮

41     }

43     return 0;
44 }
```

Growable, alokacja pamięci, kont.

- Alokacja:

mem-grow.c (28–40)

```
28          char *tmp = vmalloc(capacity);
29          if ( unlikely (!tmp))
30              return -ENOMEM;

32          printk(KERN_INFO "mem: new capacity %zu\n", capacity);

34          if (memory) {
35              memcpy(tmp, memory, mem_size);
36              vfree(memory);
37          }

39          memory = tmp;
40          mem_capacity = capacity;
```

Growable, testowanie

- `cat mem-grow.ko >/dev/mymem`
- `cat /dev/mymem >foo`
- `cmp mem-grow.ko foo`

Synchronised

- W poprzednim sterowniku występował wyścig.
- Trzeba dodać mechanizm synchronizujący chroniący dostęp do bufora.
- Mutex nadaje się idealnie.
- Funkcjonalność modułu taka jak poprzedniej wersji.

Synchronised, deklaracje

- Dodany nagłówek potrzebny do korzystania z muteksów:

`mem-sync.c` (6–15)

```
6  #include <linux/module.h>
7  #include <linux/miscdevice.h>
8  #include <linux/fs.h>
9  #include <linux/uaccess.h>
10 #include <linux/vmalloc.h>
11 #include <linux/mutex.h>

13 MODULE_DESCRIPTION("Example growable 'memory' driver with
       synchronisation");
14 MODULE_AUTHOR("Michal Nazarewicz");
15 MODULE_LICENSE("GPL");
```

Synchronised, definicja muteksa

- Dodajemy mutex:

mem-sync.c (18–23)

```
18 static DEFINE_MUTEX(mem_mutex);

20 #define mem_max_size (size_t)(1 << 20)    /* mult. of PAGE_SIZE */
21 static size_t mem_capacity;
22 static size_t mem_size;
23 static char *memory;                       /* guarded by mutex */
```

Synchronised, operacja `read(2)`

- Blokowanie przy odczycie:

`mem-sync.c` (73–87)

```
73      if (*offp >= mem_size || !len)
74          return 0;

76      if ( unlikely ( mutex_lock_interruptible (&mem_mutex)))
77          return -EINTR;

79      len = min(len, ( size_t )(mem_size - *offp));
80      ret = len;
81      if ( unlikely ( __copy_to_user (buf, memory + *offp, len)))
82          ret = -EFAULT;
83      else
84          *offp += len;

86      mutex_unlock(&mem_mutex);
87      return ret;
```

Synchronised, operacja `write(2)`

- Blokowanie przy zapisie:

`mem-sync.c` (104–122)

```
104     if ( unlikely ( mutex_lock_interruptible (&mem_mutex)))
105         return -EINTR;

107     ret = __ensure_capacity(min((size_t)(*offp + len), (size_t)mem_max_size));
108     if ( unlikely (ret < 0))
109         goto done;

111     len = min(len, (size_t)(mem_capacity - *offp));
112     ret = -EFAULT;
113     if ( likely (!__copy_from_user(memory + *offp, buf, len))) {
114         ret = len;
115         *offp += len;
116         mem_size = max((size_t)*offp, mem_size);
117         printk(KERN_INFO "mem: new size %zu\n", mem_size);
118     }

120 done:
121     mutex_unlock(&mem_mutex);
122     return ret;
```

Parametrised

- Parametryzowany rozmiar maksymalny (`max_size`).
- Możliwość zmiany rozmiaru danych (`size`).
- Możliwość zmiany rozmiaru bufora (`capacity`).
- Dopuszczamy sytuację `size` $>$ `max_size`, gdy zmniejszamy limit.

Parametrised, definicje

- `mem_max_size` to teraz zmienna, a nie makro.
- Dodane `mem_max_size_limit` (1 GiB).
- Typ zmieniony na `unsigned`.

mem-param.c (18–24)

```
18 static DEFINE_MUTEX(mem_mutex);

20 #define mem_max_size_limit (unsigned)(1 << 30)
21 static unsigned mem_max_size = 1 << 20; /* not guarded by mutex */
22 static unsigned mem_capacity;           /* at times read w/o mutex */
23 static unsigned mem_size;               /* ditto */
24 static char *memory;
```

Parametrised, warunek w sekcji krytycznej

- W funkcjach `mem_read()` warunki sprawdzane przed sekcją krytyczną są sprawdzane również wewnątrz:

`mem-param.c` (153–162)

```
153     if (*offp >= mem_size || !len)
154         return 0;

156     if ( unlikely ( mutex_lock_interruptible (&mem_mutex)))
157         return -EINTR;

159     if (*offp >= mem_size) {
160         ret = 0;
161         goto done;
162     }
```

Parametrised, warunek w sekcji krytycznej, kont.

- Analogicznie w funkcjach `mem_write()`.
- Dodatkowo zabezpieczamy się przed sytuacją, gdy `size` > `max_size`:

`mem-param.c` (188–204)

```
188     if (*offp >= max(mem_size, mem_max_size))
189         return -ENOSPC;

191     if ( unlikely ( mutex_lock_interruptible (&mem_mutex)))
192         return -EINTR;

194     limit = max(mem_size, mem_max_size);
195     if (*offp >= limit) {
196         ret = -ENOSPC;
197         goto done;
198     }

200     ret = __ensure_capacity (min((size_t)(*offp + len), limit), 0);
201     if ( unlikely (ret < 0))
202         goto done;

204     len = min(len, (size_t)(limit - *offp));
```

Parametrised, obsługa parametrów

- Parametry dostarczane jako tekst trzeba parsować:

mem-param.c (51–63)

```
51 static int mem_parse(const char *val, int min, int max)
52 {
53     unsigned long long num;
54     char *endp;
55     num = memparse(val, &endp);
56     if (endp == val)
57         return -EINVAL;
58     if (num > max || num < min)
59         return -EDOM;
60     if (*endp == '\n')
61         ++endp;
62     return *endp ? -EINVAL : num;
63 }
```

Parametrised, definiowanie parametrów

- Parametru modułu deklarujemy za pomocą jednej z deklaracji `module_param*()`:

mem-param.c (122–127)

```
122 module_param_call(maxsize, param_set_maxsize, param_get_uint,
123                   &mem_max_size, 0644);
124 module_param_call(size, param_set_size, param_get_uint,
125                   &mem_size, 0644);
126 module_param_call(capacity, param_set_capacity, param_get_uint,
127                   &mem_capacity, 0644);
```

Parametrised, ustawianie limitu danych

- Funkcja zmienia maksymalny rozmiar danych.
- Możliwe, że maksymalny rozmiar będzie mniejszy od aktualnego.
- Zmiana limitu nie wymaga zajęcia muteksa:

mem-param.c (66–75)

```
66 static int param_set_maxsize(const char *val, struct kernel_param *kp)
67 {
68     int ret = mem_parse(val, 0, mem_max_size_limit);
69     if (unlikely(ret < 0))
70         return ret;
72     mem_max_size = ret;
74     return 0;
75 }
```

Parametrised, ustawianie rozmiaru danych

- Funkcja zmieniająca rozmiaru danych.
- Możliwe zwiększenie wypełniając zerami:

mem-param.c (78–103)

```
78  static int param_set_size(const char *val, struct kernel_param *kp)
79  {
80      int ret = mem_parse(val, 0, max(mem_size, mem_max_size));
81      if (unlikely(ret < 0))
82          return ret;

84      if (unlikely(mutex_lock_interruptible(&mem_mutex)))
85          return -EINTR;

        ⋮

101     mutex_unlock(&mem_mutex);
102     return ret;
103 }
```

Parametrised, ustawianie rozmiaru danych, kont.

- Zmiana rozmiaru:

mem-param.c (87–99)

```
87      if (ret <= mem_size) {
88          mem_size = ret;
89          ret = 0;
90      } else if (ret > mem_max_size) {
91          ret = -EDOM;
92      } else {
93          int size = ret;
94          ret = __ensure_capacity(size, 0);
95          if (likely(ret >= 0)) {
96              memset(memory + mem_size, 0, size - mem_size);
97              mem_size = size;
98          }
99      }
```

Parametrised, ustawianie rozmiaru bufora

- Funkcja zmieniająca rozmiaru bufora (ale nie mniej niż `mem_size`):

`mem-param.c` (106–119)

```
106 static int param_set_capacity(const char *val, struct kernel_param *kp)
107 {
108     int ret = mem_parse(val, 0, mem_max_size_limit);
109     if (unlikely(ret < 0))
110         return ret;
112     if (unlikely(mutex_lock_interruptible(&mem_mutex)))
113         return -EINTR;
115     ret = __ensure_capacity(max((unsigned)ret, mem_size), 1);
117     mutex_unlock(&mem_mutex);
118     return ret;
119 }
```

Parametrised, ustawianie rozmiaru bufora, kont.

mem-param.c (27–48)

```
27  static int __ensure_capacity(unsigned capacity, int force)
28  {
29      capacity = (capacity + ~PAGE_MASK) & PAGE_MASK;

31      if (mem_capacity < capacity || (force && capacity != mem_capacity)) {
32          char *tmp = NULL;
33          if (unlikely(capacity)) {
34              tmp = vmalloc(capacity);
35              if (unlikely(!tmp))
36                  return -ENOMEM;
37              if (memory) {
38                  memcpy(tmp, memory, mem_size);
39                  vfree(memory);
40              }
41          }

43          memory = tmp;
44          mem_capacity = capacity;
45      }

47      return 0;
48  }
```

Parametrised, testowanie

- `insmod mem-param.ko maxsize=16K`
- `cat /sys/module/mem_param/parameters/*`
- `cat mem-param.ko >/dev/mymem`
- `cat /dev/mymem >foo; cmp mem-param.ko foo`
- `cat /sys/module/mem_param/parameters/*`
- `cat mem-param.c >/dev/mymem`
- `echo 1K >/sys/module/mem_param/parameters/size`
- `cat /dev/mymem`

O czym była prezentacja?

- Przedstawiona struktura sterownika – inicjacja i czyszczenie.
- Pokazana implementacja prostych operacji plikowych.
- Zarządzanie pamięcią przy wykorzystaniu `vmalloc()`.
- Podstawy synchronizacji – wzajemne wykluczanie przy pomocy muteksa.

- Jak widać, to nie jest aż tak trudne...
- Idzie zrozumieć. :)

Czego zabrakło?

- Dane prywatne plików (a nie globalne zmienne).
- Inne operacje plikowe – `mmap(2)`, `ioctl(2)`, ...
- Tryb nieblokujący, operacja `poll(2)`.
- Wykorzystanie `kmalloc()` czy prywatnych pól pamięci.
- Inne sposoby synchronizacji – zmienne atomowe, *spinlocki*, liczniki referencji.
- Proste struktury danych – listy, drzewa czerwono-czarne.

A to tylko wierzchołek góry lodowej...

- Komunikacja ze sprzętem.
- Przerwania.
- Górne i dolne połówki.
- DMA.
- Urządzenia blokowe.
- Systemy plików.
- Dziesiątki podsystemów.

- Testując można sobie popsuć sprzęt!

- Chyba nie ma osoby, która rozumie to wszystko...

Skąd czerpać informacje

- Use the source Luke!
- Podkatalog `Documentation` w źródłach jądra.
- Publiczne serwery LXR, np.: http://lxr.linux.no/.
- http://lwn.net/
- *Linux Device Drivers, Third Edition* udostępniony przez O'Reilly: http://lwn.net/Kernel/LDD3/.
- Daniel P. Bovet, Marco Cesati. *Understanding the Linux Kernel*

Dziękuję za uwagę!

- Pytania?
- Opinie?
- Sugestie?

- Michał “mina86” Nazarewicz
- http://mina86.com/
- mina86@mina86.com
- mina86@jabber.org

Prawnik mi kazał

- Wszystkie fragmentu kodu źrodłowego dostępna na zasadach licencji:
  - GNU General Public Licence wersji 2 lub dowolnej nowszej; lub
  - nowej licencji BSD.
- Prezentacja dostępna na zasadach licencji
  - Creative Commons Uznanie autorstwa-Użycie niekomercyjne-Na tych samych warunkach 3.0 Polska; lub
  - GNU Free Documentation License wersji 1.3 lub dowolnej nowszej bez *Invariant Sections*, *Front-Cover Texts* ani *Back-Cover Texts*.
- Jeżeli potrzebna jest inna licencja proszę o kontakt.