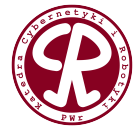
Sterowniki Robotów
Sterowniki

Wojciech Domski

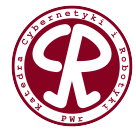
Katedra Cybernetyki i Robotyki,
Politechnika Wrocławska

Wrocław University
of Science and Technology



Plan prezentacji

Wrocław University
of Science and Technology

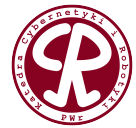


Sterownik (1/2)

Definicja sterownika urządzenia

Sterownik to most między urządzeniem a systemem operacyjnym. Wykorzystuje dobrze zdefiniowane interfejsy programistyczne zarówno do komunikacji z urządzeniem, jak i dostarczania API. Szczegóły dotyczące działania urządzenia są całkowicie ukryte.

Wrocław University of Science and Technology

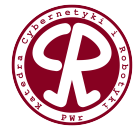


Sterownik (2/2)

Wbudowany sterownik

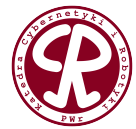
Sterownik wbudowany to interfejs, dzięki któremu programista może komunikować się z urządzeniem. Zazwyczaj dostarcza konkretne API do wymiany komunikatów i danych między programem a urządzeniem.

Wrocław University of Science and Technology



Outline

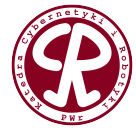
Wrocław University
of Science and Technology



Każdy sterownik powinien zawierać następujące funkcje (w zależności od typu urządzenia niektóre mogą być pominięte):

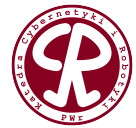
- **inicjalizacja** – inicjuje urządzenie i przygotowuje je do dalszej pracy. Po procesie inicjalizacji urządzenie powinno być gotowe do normalnej pracy i nie wymagać dodatkowych kroków do dalszych działań (konfiguracji).
- **deinicjalizacja** – zamyka urządzenie. Wykonywanie dodatkowych działań na urządzeniu nie powinno się odbywać.

Wrocław University of Science and Technology



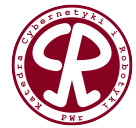
- **operacje odczytu/zapisu** – implementuje operacje odczytu/zapisu przeznaczone dla konkretnych urządzeń. Takie urządzenia charakteryzują się gotowością do działania zaraz po wykonaniu inicjalizacji i mogą być traktowane jako urządzenia typu taśmowego.
- **operacje kontrolne** – umożliwiają wykonywanie określonych funkcji na urządzeniu, np. zmianę jego stanu, wysyłanie komend, itp. W zależności od urządzenia, mogą zostać zaimplementowane różne funkcje dla różnych operacji.

Wrocław University of Science and Technology



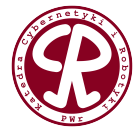
Urządzenia znakowe (z ang. *character devices*) systemu Linux korzystają z tego typu architektury. Wyżej wymienione funkcje są implementowane przy użyciu wywowałań systemowych (z ang. *system calls*) takich jak `open()`/`close()`, `read()`/`write()` oraz `ioctl()`.

Wrocław University of Science and Technology



Outline

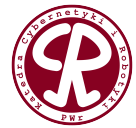
Wrocław University
of Science and Technology



Typy implementacji (1/2)

- Implementacja bezpośrednia
- Logika z separacją warstwy sprzętowej
- Wiele warstw abstrakcji
- Implementacja sterownika odwzorowująca wywołania systemowe

Wrocław University
of Science and Technology

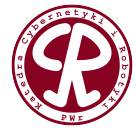


Typy implementacji (2/2)

Aspekty implementacyjne

- przenośność
- utrzymywalność
- rozszerzalność

Wrocław University
of Science and Technology



Reprezentacja urządzenia

W celu umożliwienia wielokrotnej obecności urządzeń w systemie stan pojedynczego urządzenia powinien być oddzielony od innych urządzeń. Dlatego też stan urządzenia powinien być enkapsulowany. Można to osiągnąć poprzez stworzenie dedykowanej struktury, która przechowuje wszystkie potrzebne elementy, w tym interfejsy, przez które warstwa wyższa może komunikować się z urządzeniem.

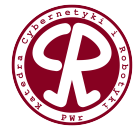
Programowanie obiektowe

Ten koncept jest obecny w językach OOP. Jednym z nich jest C++. Jednak język C może dostarczyć podobnej funkcjonalności przy użyciu struktur i wskaźników do funkcji, jeśli to konieczne.

Wrocław University of Science and Technology

Outline

Wrocław University
of Science and Technology

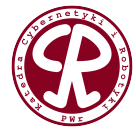


Implementacja bezpośrednia

Ten rodzaj implementacji sterownika charakteryzuje się brakiem separacji interfejsów zależnych od sprzętu. Dlatego też przenośność jest bardzo niska i wymaga wiele wysiłku, aby przenieść sterownik na inną platformę.
Co więcej, jeśli biblioteka używana do implementacji interfejsu sprzętowego zmienia się, to sterownik wymaga dokładnej uwagi przy dokonywaniu odpowiednich zmian.

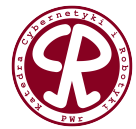
W tego typu implementacji zawieranie urządzenia jest bardzo rzadkie i zaniedbane.

Wrocław University of Science and Technology



Rozważmy przykład, w którym istnieje sterownik dla akcelerometru. Bezpośrednia implementacja założyłaby inicjalizację i deinicjalizację urządzenia oraz odczytywanie przyspieszeń z urządzenia.

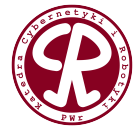
Wrocław University of Science and Technology



Implementacja

```
1   void init(void){
2     //inicjalizacja urządzenia
3   }
4
5   void deinit(void){
6     //deinicjalizacja urządzenia
7   }
8
9   void read_accel(uint16_t * data){
10    uint16_t raw_data[6];
11
12    HAL_I2C_read(raw_data);
13
14    //przetwarzanie odczytanych danych
15  }
```

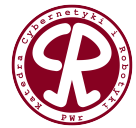
Wrocław University
of Science and Technology



Programowa reprezentacja urządzenia

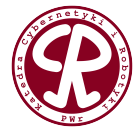
```
1    typedef struct{
2      int resolution;
3      int axis[3];
4      // ...
5    } accel_t;
```

Wrocław University of Science and Technology



Outline

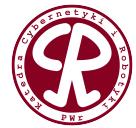
Wrocław University
of Science and Technology



Logic with hardware abstraction layer

Interfejs sprzętowy jest trzymany oddzielnie od logiki sterownika. Ten rodzaj implementacji charakteryzuje się wysoką przenośnością, ponieważ tylko warstwa sprzętowa jest dotknięta, gdy kod jest przenoszony na inne urządzenie. Czasami podwójna warstwa abstrakcji sprzętu jest implementowana, jeśli urządzenie wspiera różne interfejsy komunikacyjne. Na przykład, akcelerometr z dwoma interfejsami: I2C i SPI.

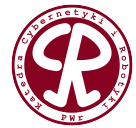
Wrocław University of Science and Technology



Przykład

Rozważmy przykład, w którym istnieje sterownik dla akcelerometru. Sterownik powinien posiadać dwa rodzaje funkcji. Pierwsza grupa funkcji powinna być traktowana jedynie jako otoczka (warstwa sprzętowa) dla interfejsu sprzętowego, podczas gdy druga grupa implementuje logikę.

Wrocław University of Science and Technology



Pseudokod warstwy sprzętowej

```
1   void read_hw(uint16_t * data, uint16_t len){
2     HAL_I2C_read(data, len);
3   }
4
5   void write_hw(uint16_t * data, uint16_t len){
6     HAL_I2C_write(data, len);
7   }
```

Wrocław University
of Science and Technology

Pseudokod warstwy logicznej

```
1   void init(void){
2     //inicjalizacja urządzenia
3     write_hw();
4   }
5
6   void deinit(void){
7     //inicjalizacja urządzenia
8     write_hw();
9   }
10
11  void read_accel(uint16_t * data){
12    uint16_t raw_data[6];
13
14    //sprzętowo zależne wywolanie
15    write_hw(raw_data);
16
17    //przetwarzanie danych
18  }
```

Wrocław University of Science and Technology

Programowa reprezentacja urządzenia

Warstwa sprzętowa:

```
1    typedef struct{
2      void * interface;
3      void * gpio;
4      //...
5    } accel_hw_t;
```

Warstwa logiczna:

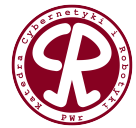
```
1    typedef struct{
2      // warstwa sprzętowa
3      accel_hw_t * hw;
4
5      // stan urządzenia
6      int resolution;
7      int axis[3];
8      //...
9    } accel_t;
```

Wrocław University of Science and Technology

Struktura katalogu

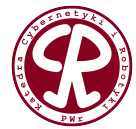
- src
  - accel.c
  - accel_hw.c
- inc
  - accel.h
  - accel_hw.h

Wrocław University
of Science and Technology



Outline

Wrocław University
of Science and Technology



Wiele warstw abstrakcji

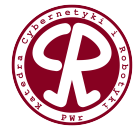
Ten rodzaj implementacji sterownika charakteryzuje się więcej niż dwoma warstwami abstrakcji. Zazwyczaj obecne są dwie warstwy – warstwa logiczna i warstwa sprzętowa. Dodatkowa warstwa może być dodana w celu dalszej generalizacji dostępu do urządzenia poprzez implementację jednostki, która reprezentowałaby samo (ogólne) urządzenie, a następnie konkretne urządzenie, i ostatecznie warstwę sprzętową do komunikacji z urządzeniem.

Wrocław University of Science and Technology

Przykład

Istnieją dwa urządzenia – akcelerometr i żyroskop. Oba urządzenia są urządzeniami typu MEMS i mogą być uważane za komponenty IMU.
Najpierw tworzony jest ogólny sterownik urządzenia, a następnie na jego podstawie tworzone są dwa sterowniki do konkretnych zastosowań.

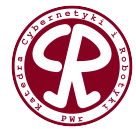
Wrocław University of Science and Technology



Sprzętowa warstwa abstrakcji

```
1  void read_hw(uint16_t * data, uint16_t len){
2    HAL_I2C_read(data, len);
3  }
4
5  void write_hw(uint16_t * data, uint16_t len){
6    HAL_I2C_write(data, len);
7  }
```

Wrocław University of Science and Technology



Warstwa abstrakcji konkretnego urządzenia

Akcelerometr:

```
1  void accel_init(void){
2  }
3
4  void accel_deinit(void){
5  }
6
7  void accel_read(uint16_t * data){
8  }
```

Żyroskop:

```
1  void gyro_init(void){
2  }
3
4  void gyro_deinit(void){
5  }
6
7  void gyro_read(uint16_t * data){
8  }
```

of Science and Technology

Warstwa abstrakcji urządzenia

```
1   void dev_init(int type){
2     if(type == 0){
3       accel_init();
4     } else {
5       gyro_init();
6     }
7   }
8
9   void dev_deinit(int type){
10    if(type == ...){
11      //deinicjalizacja konkretnego urządzenia
12    }
13  }
14
15  void dev_read(int type, uint16_t * data){
16    if(type == ...){
17      //odczyt danych
18    }
19  }
```

Wrocław University of Science and Technology

Programowa reprezentacja urządzenia (1/3)

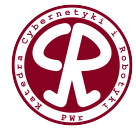
Warstwa sprzętowa (akcelerometr):

```
1    typedef struct{
2      void * interface;
3      void * gpio;
4      //...
5    } accel_hw_t;
```

Warstwa sprzętowa (żyroskop):

```
1    typedef struct{
2      void * interface;
3      void * interrupt;
4      //...
5    } gyro_hw_t;
```

Wrocław University of Science and Technology



Programowa reprezentacja urządzenia (2/3)

Warstwa logiczna (akcelerometr):

```
1   typedef struct{
2     //warstwa sprzętowa
3     accel_hw_t * hw;
4
5     //stan urządzenia
6     int resolution;
7     int axis[3];
8     //...
9   } accel_t;
```

Warstwa logiczna (żyroskop):

```
1   typedef struct{
2     //warstwa sprzętowa
3     gyro_hw_t * hw;
4
5     //stan urządzenia
6     int range;
7     int axis[3];
8     //...
9   } gyro_t;
```

Programowa reprezentacja urządzenia (3/3)

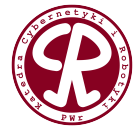
Warstwa urządzenia:

```
1   typedef struct{
2     int type;
3     void * device;
4     //...
5     int (*init)(void * dev, void * hw, void * data);
6     //...
7   } device_t;
```

Wrocław University of Science and Technology

Outline

Wrocław University
of Science and Technology



Implementacja sterownika odwzorowująca wywołania systemowe

Główną ideą jest model sterownika urządzenia w systemie Linux. Dostęp do urządzenia jest realizowany poprzez implementację standardowych wywołań systemowych [**?**]. Jednakże, ta implementacja jest możliwa na „pełnych” systemach operacyjnych, co oznacza, że pełne wykorzystanie jej możliwości może nie być w pełni możliwe bez dodatkowego nakładu pracy.
Model zakłada (między innymi) istnienie następujących operacji (wywołań systemowych):

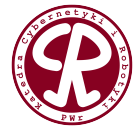
- `open()`
- `close()`
- `read()`
- `write()`
- `ioctl()`

Wrocław University of Science and Technology

Prototypy wywołań systemowych

```
1       int (*open) (struct inode *, struct file *);
2       int (*release) (struct inode *, struct file *);
3
4       ssize_t (*read) (struct file *, char __user *, size_t,
            loff_t *);
5       ssize_t (*write) (struct file *, const char __user *, size_t
            , loff_t *);
6
7       int (*ioctl) (struct inode *, struct file *, unsigned int,
            unsigned long);
```

Wrocław University of Science and Technology



Przykład

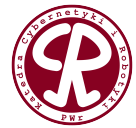
Rozważmy przykład, w którym istnieje sterownik dla akcelerometru i żyroskopu. Rozróżnienie między dwoma lub większą liczbą typów urządzeń może być dokonane na poziomie funkcji "wywołań systemowych".

Wrocław University of Science and Technology

Sprzętowa warstwa abstrakcji

```
1  void read_hw(uint16_t * data, uint16_t len){
2    HAL_I2C_read(data, len);
3  }
4
5  void write_hw(uint16_t * data, uint16_t len){
6    HAL_I2C_write(data, len);
7  }
```

Wrocław University
of Science and Technology



Warstwa abstrakcji konkretnego urządzenia

Akcelerometr:

```
1  void accel_init(void){
2  }
3
4  void accel_deinit(void){
5  }
6
7  void accel_read(uint16_t * data){
8  }
```

Żyroskop:

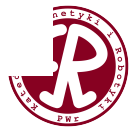
```
1  void gyro_init(void){
2  }
3
4  void gyro_deinit(void){
5  }
6
7  void gyro_read(uint16_t * data){
8  }
```

of Science and Technology

Abstrakcja urządzenia

```
1   int open(struct inode *, struct file *){
2     // inicjalizacja urządzenia
3   }
4   int release(struct inode *, struct file *){
5     // deinicjalizacja urządzenia
6   }
7
8   ssize_t read(struct file *, char __user *, size_t, loff_t *)
        {
9     // odczyt danych z urządzenia
10  }
11  ssize_t write(struct file *, const char __user *, size_t,
        loff_t *){
12    // zapis danych do urządzenia
13  }
14
15  int ioctl(struct inode *, struct file *, unsigned int,
        unsigned long){
16    // sterowanie urządzeniem
17  }
```

of Science and Technology



Programowania reprezentacja urządzenia (1/3)

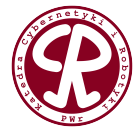
Warstwa sprzętowa (akcelerometr):

```
1  typedef struct{
2    void * interface;
3    void * gpio;
4    //...
5  } accel_hw_t;
```

Warstwa sprzętowa (żyroskop):

```
1  typedef struct{
2    void * interface;
3    void * interrupt;
4    //...
5  } gyro_hw_t;
```

Wrocław University of Science and Technology

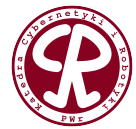


Programowania reprezentacja urządzenia (2/3)

Warstwa urządzenia (akcelerometr):

```
1   typedef struct{
2     //warstwa sprzętowa
3     accel_hw_t * hw;
4
5     //stan urządzenia
6     int resolution;
7     int axis[3];
8
9     //operacje, otwarcie urządzenia, odczyt, ...
10    int (*open) (struct inode *, struct file *);
11    //...
12  } accel_t;
```

Wrocław University of Science and Technology

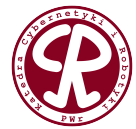


Programowania reprezentacja urządzenia (3/3)

Warstwa urządzenia (żyroskop):

```
1   typedef struct{
2     //warstwa sprzętowa
3     gyro_hw_t * hw;
4
5     //stan urządzenia
6     int range;
7     int axis[3];
8
9     //operacje, otwarcie urządzenia, odczyt, ...
10    int (*open) (struct inode *, struct file *);
11    //...
12  } gyro_t;
```

Wrocław University of Science and Technology



Outline

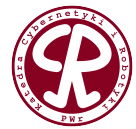
Wrocław University
of Science and Technology

Techniques

Jednym z najważniejszych aspektów podczas implementacji sterownika jest wybór metody separacji zależności sprzętowych od warstwy logicznej. Może to być wykonane przy użyciu różnych technik.

- definicje, kod wybierany na etapie kompilacji,
- dodatkowe funkcje z przyrostkiem `_hw`,
- funkcje `__weak`,
- wskaźniki do funkcji,
- klasy abstrakcyjne.

Wrocław University of Science and Technology



Optymalizacja bez optymalizacji

Z uwagi na wykorzystywanie rozmaitych bibliotek na potrzeby wytwarzania oprogramowania wbudowanego zachodzi potrzeba optymalizacji w celu zredukowania kodu maszynowego. W tym celu można wykorzystać optymalizację dostępna za pośrednictwem kompilatora i przełączników `-O`. Natomiast możliwe jest ograniczenie wielkości kodu wynikowego przez usunięcie nieużywanych sekcji – zarówno funkcji jak i danych.

1. `-fdata-sections`, `-ffunction-sections`,
2. `-Wl`, `-gc-sections`.

Wrocław University of Science and Technology