Programmazione a oggetti in C nel kernel di Linux

Luca Ceresoli

Introduzione

Programmazione non orientata agli oggetti

- Il software è un insieme di procedure che manipolano dati
- Le procedure accedono ai dati direttamente

Esempio: shopping cart

```
struct Item {
  string name;
  int    price;
};

int main() {
  list<Item> cart1, cart2;
  Item kr = {"The C Programming Language", 53};
  Item bs = {"The C++ Programming Language", 56};
  Item bc = {"Understanding the Linux Kernel", 45};
  cart1.push_back(kr);
  cart1.push_back(bs);
  cart2.push_back(kr);
  cart2.push_back(bc);
  int total1 = 0, total2 = 0;
  for (Item &i: cart1)
    total1 += i.price;
  for (Item &i: cart2)
    total2 += i.price;
  cout << "Cart1 total: " << total1 << endl;
  cout << "Cart2 total: " << total2 << endl;
}
```

Programmazione orientata agli oggetti

- Ogni entità logica è rappresentata da un oggetto
- Un oggetto contiene:
  - *proprietà*: i dati che la definiscono
  - *metodi*: possibili azioni che agiscono sui dati
- Classe:
  - descrizione astratta di oggetti dello stesso tipo

Esempio ad oggetti /1

```
struct Item {
  string name;
  int    price;
};

class ShoppingCart {
  public:
    void   add(Item &item);
    int    getTotal();
  private:
    list<Item> items;
};

void ShoppingCart::add(Item &item) {
  items.push_back(item);
}

int ShoppingCart::getTotal() {
  int total = 0;
  for (Item &i: items)
    total += i.price;
  return total;
}
```

Esempio ad oggetti /2

```
int main()
{
  ShoppingCart cart1, cart2;
  Item kr = {"The C Programming Language", 53};
  Item bs = {"The C++ Programming Language", 56};
  Item bc = {"Understanding the Linux Kernel", 45};
  cart1.add(kr);
  cart1.add(bs);
  cart2.add(kr);
  cart2.add(bc);
  cout << "Cart1 total: " << cart1.getTotal() << endl;
  cout << "Cart2 total: " << cart2.getTotal() << endl;
}
```

- Un linguaggio a oggetti (C++) offre molti vantaggi
- Ma molti software sono tuttora scritti in C
  - Software semplici
    - Firmware per microcontrollori
    - Utility di dimensioni contenute
  - Ma anche software molto complessi (e strutturati a oggetti)
    - Linux kernel
    - GTK+
    - EFL

Classi e oggetti in C

Classi e oggetti in C

- In C si possono creare delle "classi" usando le `struct`
  - Contengono le proprietà (dati), non i metodi
  - Per ricreare i metodi usiamo normali funzioni
    - Che come primo parametro ricevono un puntatore alla struct
- Proprietà e metodi non sono legati dal linguaggio, ma da una convenzione

Esempio ad oggetti in C

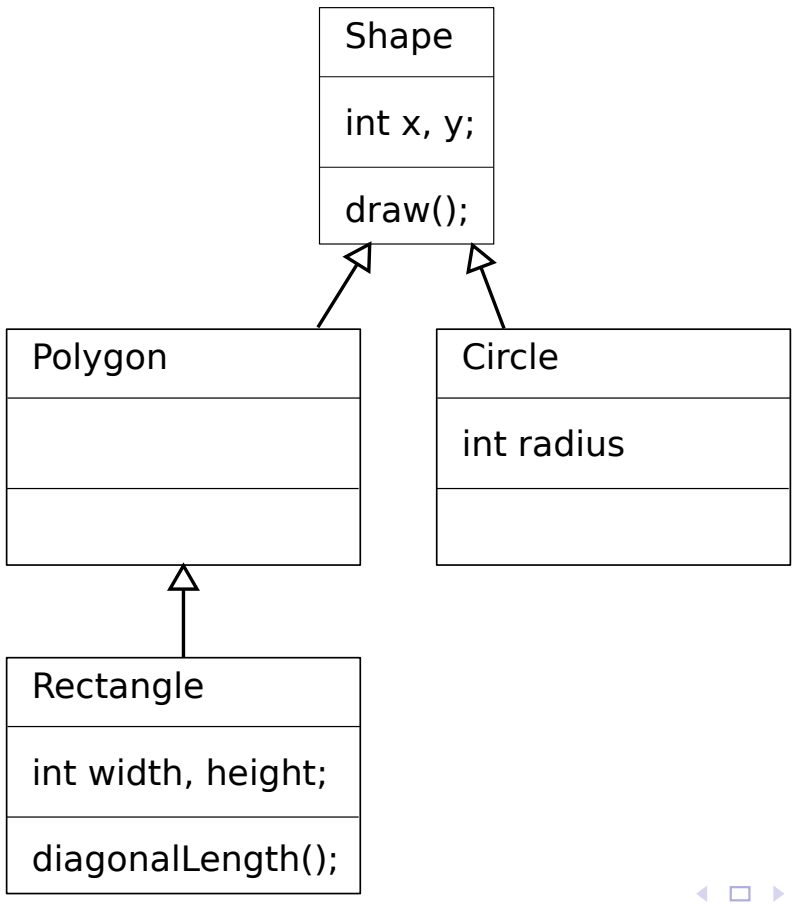
```
struct ShoppingCart {
  Item items[100];
};

int main()
{
  ShoppingCart cart1, cart2;
  shoppingcart_initialize(&cart1);
  shoppingcart_initialize(&cart2);
  Item kr = {"The C Programming Language", 53};
  Item bs = {"The C++ Programming Language", 56};
  Item bc = {"Understanding the Linux Kernel", 45};
  shoppingcart_add(&cart1, &kr);
  shoppingcart_add(&cart1, &bs);
  shoppingcart_add(&cart2, &kr);
  shoppingcart_add(&cart2, &bc);
  cout << "Cart1 total: " << shoppingcart_get_total(&cart1) << endl;
  cout << "Cart2 total: " << shoppingcart_get_total(&cart2) << endl;
}
```

Ereditarietà e polimorfismo

Ereditarietà

- Da una classe base, se ne crea un'altra (sottoclasse)
  - che ne eredità proprietà e metodi
  - e ne aggiunge di nuovi
  - o modifica il comportamento di quelli esistenti
- Si crea una gerarchia di classi



Polimorfismo

- Lo stesso metodo può agire su oggetti di classi diverse
- Ogni sottoclasse può aumentare o sostituire il metodo della classe base
- Esempio: `Shape::draw()` può essere chiamato su oggetti `Rectangle` o `Circle`

Il kernel di Linux

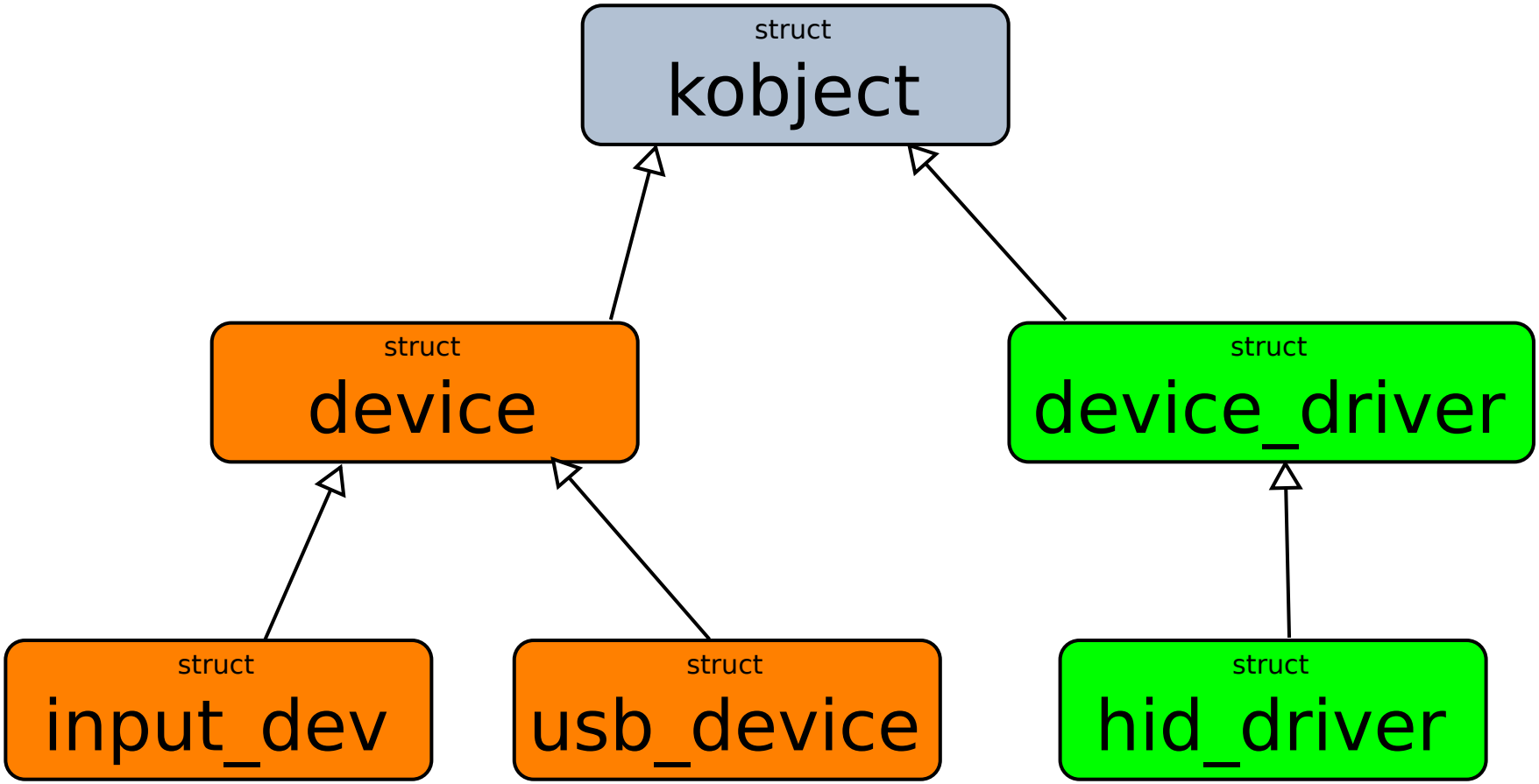
Il kernel

- È il “nocciolo” del sistema operativo
- Gestisce tutte le risorse hardware, i processi, la memoria e molto altro
- Scritto in C
- Fortemente strutturato ad oggetti
  - Il *device model*: struttura dei device e dei device drivers
  - Il *Virtual Filesystem*: gestione uniforme di diversi filesystem
  - ...

Il device model

- Modello a oggetti di tutte le periferiche del computer
- Introdotto in Linux 2.6 (2003)
- Permette:
  - Power management
  - Visibilità all'utente delle periferiche connesse
    - `sysfs`
    - `lshw`
  - Hotplug
  - Object lifecycle

II device model — class diagram



struct kobject

- Classe base da cui derivano le altre
  - Collegamenti tra oggetti
    - Esempio: un device USB punta all'hub USB cui è connesso
  - Rappresentazione in sysfs
    - Attributi del kobject diventano file in `/sys`
    - Collegamenti (puntatori) tra kobject diventano link in `/sys`
  - Generazione eventi hotplug
  - Reference counting

Ereditarietà dei dati

Ereditarierà dei dati in C++

```
class Shape
{
 private:
  float x, y;
}

class Circle : public Shape
{
 public:
  float left() {
    return (x - radius);
  }
  float right() {
    return (x + radius);
  }
 private:
  float radius;
}
```

Ereditarierà dei dati in C nel kernel

- Realizzata con diverse tecniche
  - Embedded structures (il più utilizzato)
  - Unions
  - Void pointers

Embedded structures /1

struct kobject

```
struct kobject {
  const char            *name;

  struct kobject        *parent;
  struct kernfs_node    *sd; /* sysfs directory entry */

  struct kref           kref;

  /* ... */
};
```

Source: http://lxr.free-electrons.com/source/include/linux/kobject.h?v=4.8#L63

Embedded structures /2

`struct device` eredita da `struct kobject`

```
struct device {
  struct kobject         kobj;

  struct dev_pm_info     power;
  struct dev_pm_domain  *pm_domain;

  dev_t                  devt;    /* dev_t, creates the sysfs "dev" */

  /* ... */
};
```

Source: http://lxr.free-electrons.com/source/include/linux/device.h?v=4.8#L780

`struct input_dev` eredita da `struct device`

```
struct input_dev {
  const char *name;

  unsigned long evbit[BITS_TO_LONGS(EV_CNT)];
  unsigned long keybit[BITS_TO_LONGS(KEY_CNT)];

  struct device dev;

  /* ... */
};
```

Source: http://lxr.free-electrons.com/source/include/linux/input.h?v=4.8#L121

Accesso ai dati

- Dalla classe derivata alla classe base:

```
my_input_device.dev.power
my_input_device.dev.kobj.name
```

- Dalla classe base alla classe derivata:
  - `container_of()`

La macro container_of()

- container_of(ptr, type, member)

  http://lxr.free-electrons.com/source/include/linux/kernel.h?v=4.8#L830

- Esempio: to_input_dev() dato un puntatore a struct device restituisce un puntatore alla struct input_device che lo contiene

```
struct input_dev {
  unsigned long evbit[BITS_TO_LONGS(EV_CNT)];
  unsigned long keybit[BITS_TO_LONGS(KEY_CNT)];

  struct device dev;

  /* .. */
};
#define to_input_dev(d) container_of(d, struct input_dev, dev)
```

Source: http://lxr.free-electrons.com/source/include/linux/input.h?v=4.8#L121

Ereditarietà dei metodi

vtable

- Virtual Method Table (o vtable)
  - Un insieme di puntatori a funzione
  - Ciascuno implementa un “metodo” della classe

`struct device_driver`: un generico driver HID (Human Interface Device)

```
struct hid_driver {
  char *name;

  int (*event)(struct hid_device *hdev, struct hid_field *field,
               struct hid_usage *usage, __s32 value);

  /* private: */
  struct device_driver driver;

  /*  ... */
};
```

Source: http://lxr.free-electrons.com/source/include/linux/hid.h?v=4.8#L679

struct mt_driver: un driver HID multitouch

```
static int mt_event(struct hid_device *hid, struct hid_field *field,
                    struct hid_usage *usage, __s32 value)
{
  /* ... */
}

static struct hid_driver mt_driver = {
  .name = "hid-multitouch",
  .id_table = mt_devices,

  /* ... */

  .event = mt_event,

  /* ... */
};
```

Source: http://lxr.free-electrons.com/source/drivers/hid/hid-multitouch.c?v=4.8#L1520

hid-core.c: implementazione della "classe" hid_driver

```
static void hid_process_event(struct hid_device *hid,
                              struct hid_field *field,
                              struct hid_usage *usage,
                              __s32 value, int interrupt)
{
  struct hid_driver *hdrv = hid->driver;
  /* ... */

  if (hdrv && hdrv->event && hid_match_usage(hid, usage)) {
    ret = hdrv->event(hid, field, usage, value);
    if (ret != 0) {
      if (ret < 0)
      hid_err(hid, "%s's event failed with %d\n",
      hdrv->name, ret);
      return;
    }
  }
  /* ... */
}
```

Source: http://lxr.free-electrons.com/source/drivers/hid/hid-core.c?v=4.8#L1237

Conclusioni

Conclusioni

- La programmazione a oggetti si può fare con un linguaggio non ad oggetti
- Le tecniche usate in C sono spesso quelle che i compilatori C++ usano "sotto il cofano"
- Svantaggi
  - Molto più faticosa
  - Codice più prolisso e meno leggibile
- Vantaggi
  - Più controllo su ciò che succede
  - Si possono usare tecniche differenti in base alle necessità
    - Ad esempio per ottimizzare le prestazioninei punti critici

Domande?

luca@lucaceresoli.net
http://lucaceresoli.net

Bibliografia

- Object-oriented design patterns in the kernel
  Part 1: `https://lwn.net/Articles/444910/`
  Part 2: `https://lwn.net/Articles/446317/`
- Object-oriented techniques in C
  `http://dmitryfrank.com/articles/oop_in_c`