EPFL

Architecture d'un programme interactif graphique

Partie 3: Gestion du temps

Objectifs:

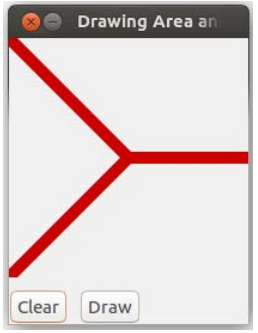
- completer la maîtrise de la boucle d’interaction

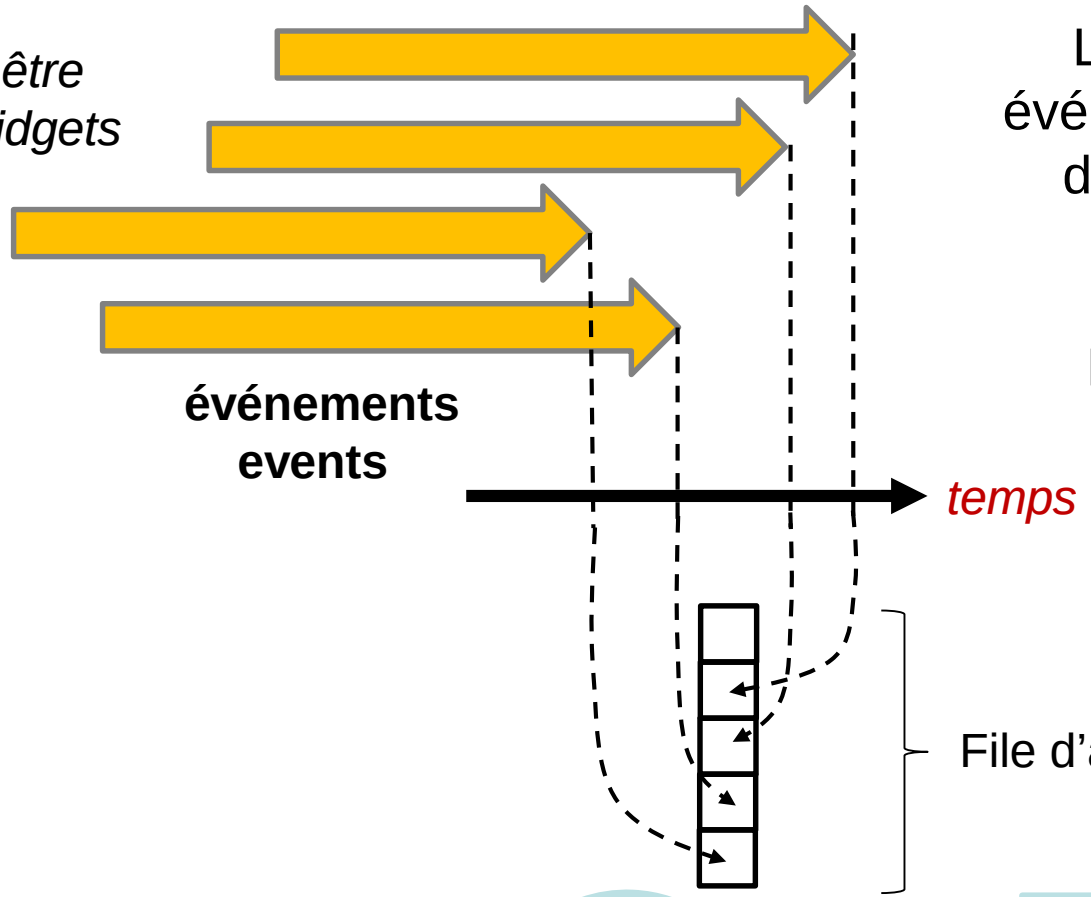
Plan:

- La file d’attente des événements
- Gestion du temps: timer, idle mode

La file d'attente des événements / FIFO = *first in first out*

Manipulation de la fenêtre
Interaction avec les widgets
Timers
+ bouton souris
+ touches clavier

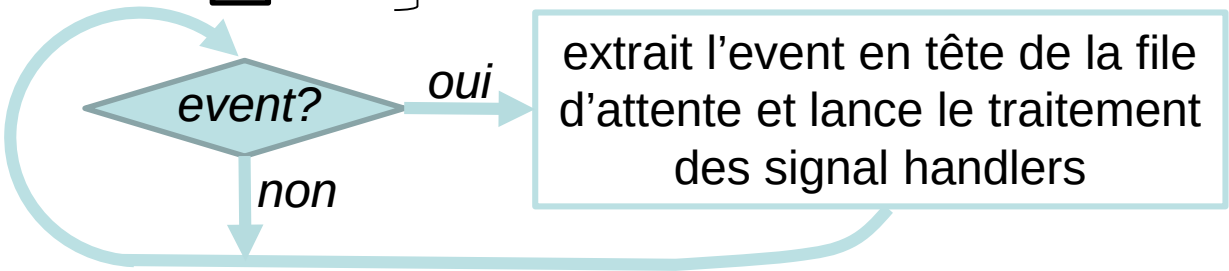




L'instant de création d'un événement détermine sa place dans ***la file d'attente des événements***:

FIFO = First In, First OUT

La méthode **run**() de l'application GTKmm exécute *une boucle inifinie* qui **vérifie** s'il y a un événement en tête de la liste d'attente.

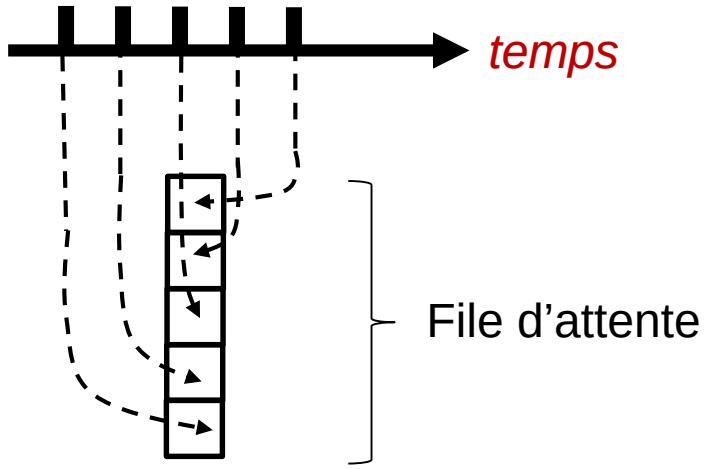


Exécuter une action à intervalles réguliers avec un Timer

Une fenêtre peut activer un (ou plusieurs) signal **timeout()**

timeout() produit un signal chaque fois d'un délai (en ms) est écoulé.

Le délai est défini au lancement du timeout



Son **signal handler** est appelé pour chaque signal émis.

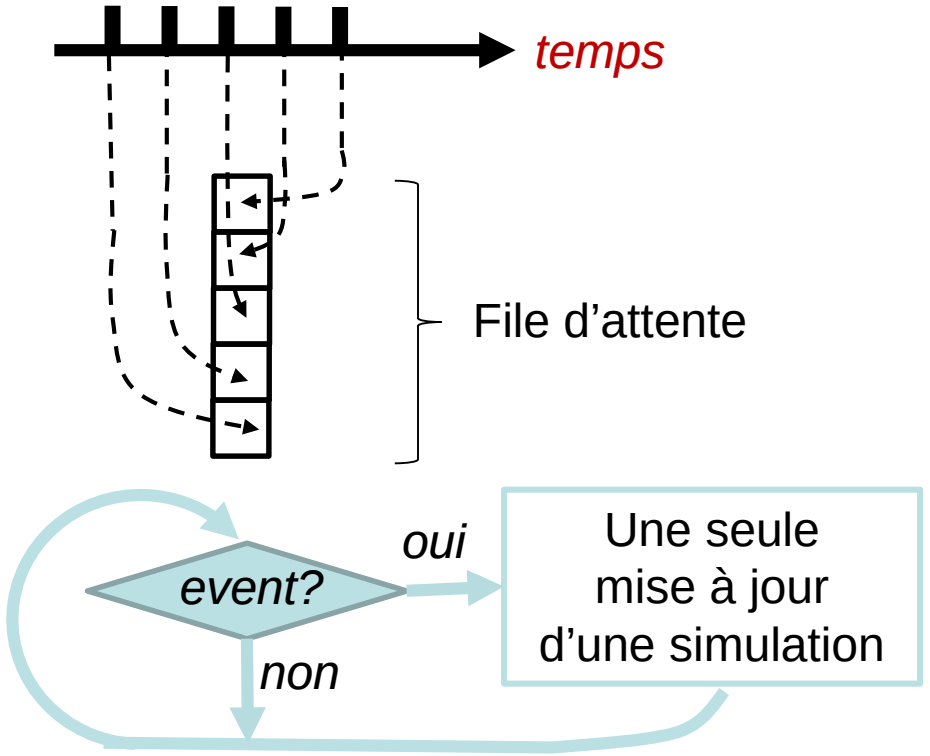
De plus, si le **signal handler** renvoie le booléen **true** : le timer **poursuit son activité**

si le **signal handler** renvoie le booléen **false** : le timer **est supprimé**

Exécuter une simulation
Sans bloquer l'interaction

L'action exécutée par un signal handler ne doit pas être trop coûteuse en temps sinon elle ne se termine pas assez rapidement et elle empêche la méthode **run** de traiter les événements de la file d'attente => elle bloque l'interaction

il faut exécuter une seule mise à jour d'une simulation avec le signal handler d'un timer



Exemple BasicTimer(1)

.h

```
…
class BasicTimer : public Gtk::Window
{
public:
  BasicTimer();

protected:
  void on_button_add_timer();
  void on_button_delete_timer();
  void on_button_quit();

  bool on_timeout();

  Gtk::Box m_Box;
  Gtk::Button m_ButtonAddTimer,
      m_ButtonDeleteTimer, m_ButtonQuit;

  bool timer_added;
  bool disconnect;

  // constant initialized in constructor
  const int timeout_value;
};…
```

.cc

```
…
BasicTimer::BasicTimer() :
  m_Box(Gtk::ORIENTATION_HORIZONTAL, 10),
  m_ButtonAddTimer("_Start", true),
  m_ButtonDeleteTimer("_Stop", true),
  m_ButtonQuit("_Quit", true),
  timer_added(false),
  disconnect(false),
  timeout_value(500) // 500 ms = 0.5 seconds
{
  set_border_width(10);

  add(m_Box);
  m_Box.pack_start(m_ButtonAddTimer);
  m_Box.pack_start(m_ButtonDeleteTimer);
  m_Box.pack_start(m_ButtonQuit);

  // Connect the three buttons:
  m_ButtonQuit.signal_clicked().connect(sigc::mem_fun(*this,
              &BasicTimer::on_button_quit));
  m_ButtonAddTimer.signal_clicked().connect(sigc::mem_fun(*this,
              &BasicTimer::on_button_add_timer));
  m_ButtonDeleteTimer.signal_clicked().connect(sigc::mem_fun(*this,
              &BasicTimer::on_button_delete_timer));

  show_all_children();
}
…
```

Exemple BasicTimer(2)

.CC

```
…
bool BasicTimer::on_timeout()
{
  static unsigned int val(1);

  if(disconnect)
  {
    disconnect = false; // reset
    return false; // End of Timer
  }

  std::cout << "This is timer value : "
            << val << std::endl;

  // tic the clock
  ++val;

  // keep the Timer working
  return true;
}
```

.CC

```
…
void BasicTimer::on_button_add_timer()
{
  if(not timer_added)
  {
    Glib::signal_timeout().connect( sigc::mem_fun(*this,
          &BasicTimer::on_timeout),timeout_value );

    timer_added = true;

    std::cout << "Timer added" << std::endl;
  }
  else
  {
    std::cout << "The timer already exists : nothing more is created"
              << std::endl;
  }
}

void BasicTimer::on_button_delete_timer()
{
  if(not timer_added)
  {
    std::cout << "Sorry, there is no active timer at the moment.“
              << std::endl;
  }
  else
  {
    std::cout << "manually disconnecting the timer "
              << std::endl;
    disconnect  = true;
    timer_added = false;
  }
}…
```

Exécuter une simulation **le plus rapidement possible**

Sans bloquer l'interaction

Mauvaise idée: donner une valeur très faible au **délai** associé au timer.

Pourquoi ? ce délai pourrait devenir insuffisant pour traiter le signal handler sur cette courte durée.

Conséquence1: accumulation d'event du timer dans la file d'attente

Conséquence2: les autres event d'interaction sont perdus dans la masse des event du timer => retard dans leur traitement

Solution: exécuter un signal handler quand il ***n'y a aucun événement*** dans la file d'attente.

Signal handler **idle**