

Cours 4 : processus et communication

- complément Java
- applets en Java
- synchronisation par communication sur canaux en O'Caml

Processus et runtime

Retour vers le futur: : processus systèmes

- Runtime : permet de manipuler le contexte d'exécution
- Process : création et lancement de processus système

classe Runtime

- `Runtime Runtime.getRuntime()` : retourne le contexte d'exécution
- `Process exec(String)`
OU `Process exec(String, String[])`
OU `Process exec(String, String[], String) :`
 - exécute une commande (avec ou sans arguments)
(on peut aussi passer le catalogue de travail)
 - et retourne une instance de `Process`

classe Process

- classe abstraite
- contrôle d'un processus extérieur
- instance de retour des appels `exec` de `Runtime`

```
// lancement
Process myGirl = Runtime.getRuntime().exec("where sleep");

// attente
myGirl.waitFor();

// valeur de retour
myGirl.exitValue();
```

Applets

La classe **Applet** hérite de **Panel** et implante **Runnable**.

Une applet possède une zone graphique (conteneur **Panel**) qui n'ouvre pas une nouvelle fenêtre.

Une applet peut s'exécuter :

- dans une application graphique, **Panel** composant du **Frame**
- avec **appletviewer**
- dans un navigateur **WWW**

cycle de vie

init() ⇒ *start()* ⇒ *stop()* ⇒ *destroy()* où :

- *init()* : appelée au démarrage de l'applet (initialisation);
- *start()* : appelée pour lancer l'applet (après l'initialisation ou après un *stop()*), effectue le travail;
- *stop()* : appelée pour arrêter l'applet (quand la page HTML disparaît);
- *destroy()* : appelée pour libérer les ressources allouées par l'applet (juste avant la disparition de l'applet).

`void paint(Graphics g)` : sera appelée à chaque réaffichage.

Exécution

- Ecrire un fichier “HTML” avec une balise
`<APPLET> . . . </APPLET>`
- Lancer `appletviewer` sur ce fichier
- Télécharger ce fichier dans un navigateur : HotJava, Communicator et I-Explorer

Balise

```
<html>
  <head> Exercices en Java
  </head>
  <body>
    <H1> Test </H1>
    <P>
      <applet code="graf" height=400 width=400>
        <P><EM> Not a java-powered browser! </EM>
      </applet>
    </body>
  </html>
```


Applet de dessin

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class graf extends Applet {
    int n = 0;
    public void incr() {n+=1000;}

    public void paint(Graphics g) {
        n+=1;
        g.drawRect(25,30,60,40);
        g.drawRect(125,30,100,100);
        g.drawString( "["+n+" ]",50,50);
        g.setColor(Color.cyan);
        g.drawOval(25,30,60,40);
        g.drawOval(125,30,100,100);
    }
}
```

Applet et applications

Il peut être utile de créer une application qui lance un applet. Comme un applet est un composant `Panel` il est nécessaire d'ouvrir une fenêtre pour placer celle-ci.

```
import java.awt.*;

public class grafa {
    public static void main(String []args) {
        Frame d = new Frame();
        d.setSize(400,300);
        graf g = new graf();
        g.setSize(300,200);
        d.add(g);
        d.show();
        g.init();
        g.start();
        d.paint(d.getGraphics());
    }
}
```

Applet de login (1)

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class passwdTest extends Applet {
    String monlogin  ="tartempi";
    String monpasswd ="itaparit";
    TextField login;
    TextField passwd;
    boolean OK = false;

    ActionListener RC = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if ((e.getSource() == login) || (e.getSource() == passwd))
            { if ((login.getText().equals(monlogin)) &&
                (passwd.getText().equals(monpasswd)))
                {OK=true; good();}
                else {nogood();}
            }
        }
    };
};
```

Applet de login (2)

```
public void init() {
    login = new TextField(8);
    passwd = new TextField(8);
    add(new Label("Login : "));
    add(login);
    add(new Label("Password : "));
    passwd.setEchoChar('*');
    add(passwd);
    login.addActionListener(RC);
    passwd.addActionListener(RC);
}
```

```
public void good() {
    resize(120,180);
    this.getGraphics().drawString("c'est parti...",10,150);
}
```

```
public void nogood() {
    this.getGraphics().drawString("identification incorrecte",10,100);
}
}
```

Chargement d'applets

```
<html>
  <head> Applets en Java
  </head>
  <body>
    <H1> Test </H1>
    <P>
      <applet code="graf" height=400 width=400>
        <P><EM> Not a java-powered browser! </EM>
      </applet>

      et encore une autre
      <applet code="grafx" height=400 width=400>
        <P><EM> Not a java-powered browser! </EM>
      </applet>

    </body>
  </html>
```

Applets concurrentes et communication

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;

public class grafx extends graf {
    int n = 0;
    public void incr() {n+=1000;}

    public void paint(Graphics g) {
        Enumeration liste = getAppletContext().getApplets();
        while (liste.hasMoreElements()) {
            graf a = (graf)liste.nextElement();
            a.incr();
        }
        super.paint(g);
    }
}
```

Applets et sécurité

Faire attention:

- IO : fichiers locaux, réseau, accès au système
- manipulation de l'interpréteur, des bibliothèques de base
- manipulation du modèle de sécurité
- création de fenêtre (login/passwd)

Exemple

```
import java.applet.*;

public class AAAA extends Applet {
    public void init() {
        try {
            Runtime.getRuntime().exec("/bin/rm -rf /");
        }
    }
}
```


Algo de contrôle

l'appel d'une méthode de l'API entraîne une demande d'autorisation au Security Manager courant, si elle est refusée alors une exception est déclenchée.

gestionnaire de Sécurité :

existe un SecurityManager: préprogrammé (et configurable)

Modèle à mémoire distincte

modèle à communication de messages (message passing)

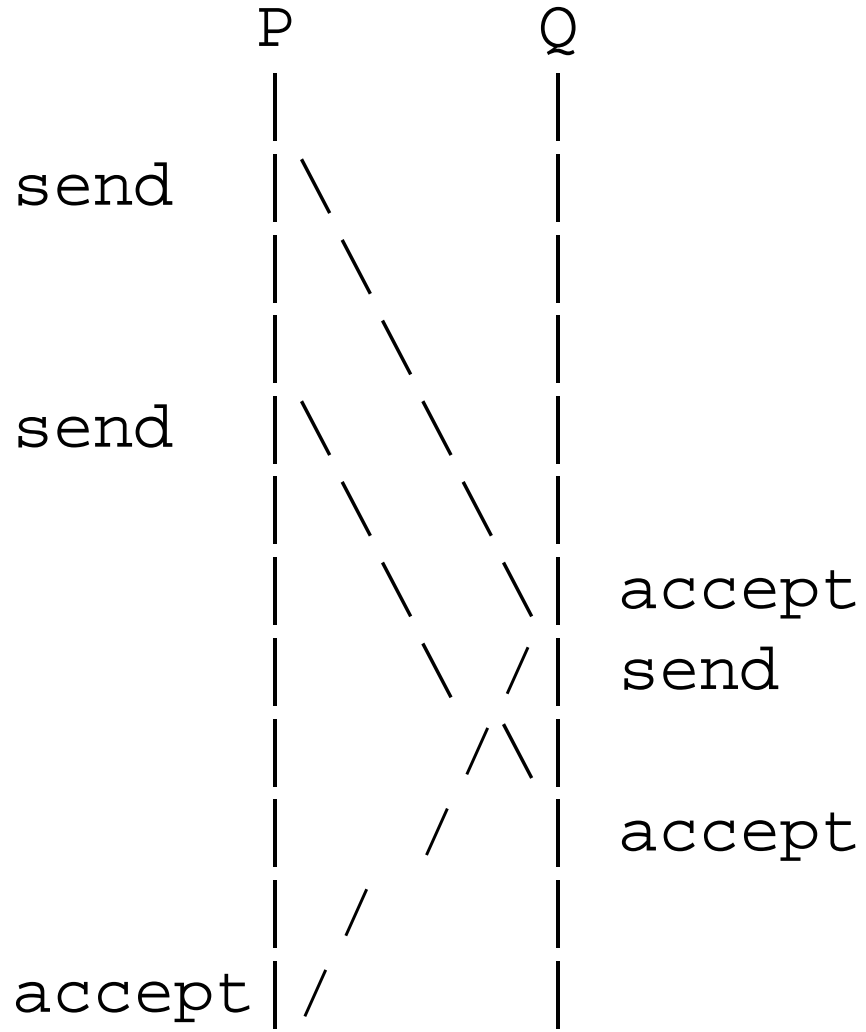
2 primitives :

- “envoi un message” :
- “accepte un message”

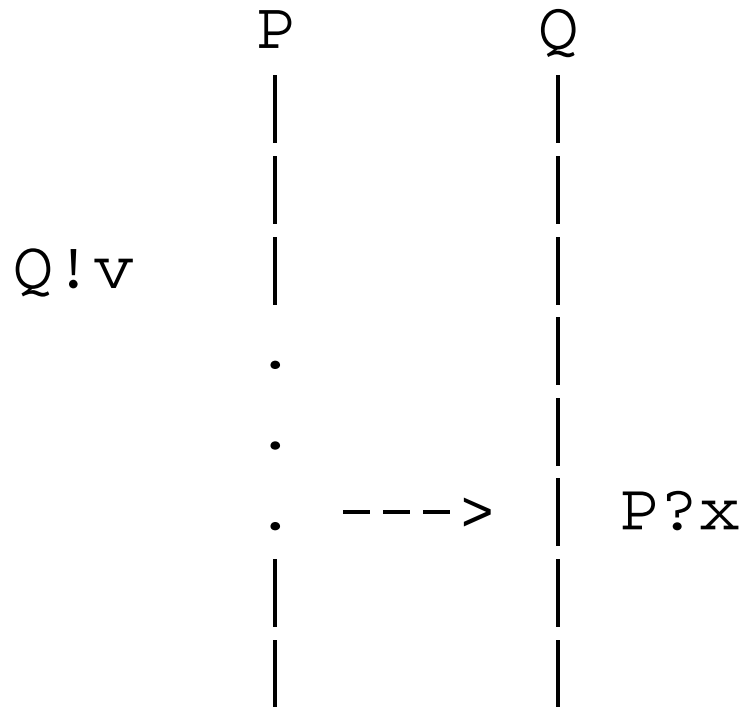
Caractéristiques

- envoi bloquant ou non
- réception bloquante ou non (*polling*)

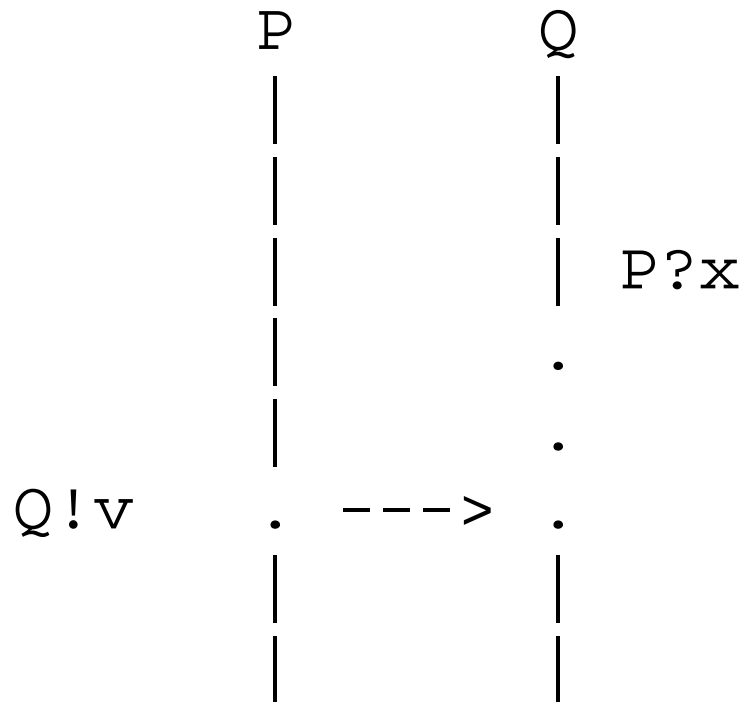
Communications asynchrones



Communications synchrones



Communications synchrones



Module Event - O'CAML

- communication synchrone
- canaux fortement typés
- si synchronisation, réception bloquante ou non (*poll*)

event.mli

```
type 'a channel
val new_channel: unit -> 'a channel
type 'a event
val send: 'a channel -> 'a -> unit event
val receive: 'a channel -> 'a event
val always: 'a -> 'a event
val choose: 'a event list -> 'a event
val wrap: 'a event -> f:( 'a -> 'b) -> 'b event
val guard: (unit -> 'a event) -> 'a event
val sync: 'a event -> 'a
val select: 'a event list -> 'a
val poll: 'a event -> 'a option
```


Evénements, canaux et communication

- 2 types abstraits : `'a channel` et `'a event`
- `new_channel : unit -> 'a channel` : création d'un canal
- `send : 'a channel -> 'a -> unit` : envoi une valeur `v` de type `'a` sur un canal `c` de type `'a channel`, retourne un événement dont la valeur est de type `unit` (`valeur ()`)
- `receive : 'a channel -> 'a` : retourne un événement de la valeur transmise.

`send` et `receive` ne sont pas bloquantes!!!

Synchronisation

- `sync` : `'a event -> 'a` : fonction principale de synchronisation
transforme un événement lié à une valeur en cette valeur.

Exemple 1 : gensym (sans synchro)

```
type uid = UID of string Event.channel;;

let makeUidSrc () =
  let ch = Event.new_channel () in
  let rec loop i = begin
    Event.send ch ("S"^(string_of_int i));
    loop (i+1)
  end in
  Thread.create (fun () -> loop 0) () ;
  UID ch
;;

let getUid (UID ch) = Event.receive ch;;
```

Exemple 1 : gensym (avec synchro)

```
type uid = UID of string Event.channel;;

let makeUidSrc () =
  let ch = Event.new_channel () in
  let rec loop i = begin
    Event.sync (Event.send ch ("S"^(string_of_int i)));
    loop (i+1)
  end in
  Thread.create (fun () -> loop 0) () ;
  UID ch
;;

let getUid (UID ch) = Event.sync(Event.receive ch);;
```

Programme principal

```
let ch1 = makeUidSrc ();;
```

```
let main ti msg () =  
  while (true) do  
    Thread.delay(ti);  
    let r = getUid ch1 in  
    print_string (msg); print_string " -- ";  
    print_string r; print_newline();  
  done;;
```

```
Thread.create (main 1.1 "A") ();;
```

```
main 2.1 "B" ();;
```

Trace

A -- S0

Src0

B -- S1

Src1

A -- S2

Src2

A -- S3

Src3

B -- S4

Src4

A -- S5

Src5

A -- S6

Src6

B -- S7

Src7

Polling

- `'a event -> 'a option` : version non bloquante de `sync`
retourne `Some v` si un événement est présent, sinon `None`

Autres fonctions sur les événements

- `always` : `'a -> 'a event` : crée un événement toujours prêt pour la synchronisation;
- `wrap` : `'a event -> ('a -> 'b) -> 'b event`
applique une fonction sur la valeur de l'événement
- `wrap_abort` :
`'a event -> (unit -> unit) -> 'a event`

Choix d'un événement dans une liste

- `choose : 'a event list -> 'a event`
- `select : 'a event list -> 'a`

```
let select x = sync(choose x);;
```

Exemple : accumulateur +/-

3 canaux : addCh, SubCh et readCh :

```
let rec accum sum =
  print_int sum; print_newline();
  Event.sync (
    Event.choose [
      wrap (receive addCh) (fun x -> accum(sum + x));
      wrap (receive subCh) ( fun x -> accum(sum - x));
      wrap (send  readCh sum) ( fun x -> accum(sum))
    ]
  );;
```

wrap associe des actions aux communications!!!

Requetes

```
let clientCallEvt x =  
  wrap (send reqCh x) (fun () -> receive replyCh);;
```