Processus concurrents et parallélisme

Chapitre 3/4/5/6 - Parallélisme/Étude de cas

Gabriel Girard

12 février 2015

Chapitre 3/4/5/6 - Parallélisme/Étude de cas

1. Langages
   - CSP
   - Ada
   - Java
   - Python
   - C#
   - C++
   - C++
   - OpenMP
   - MPI
   - MOM
2. Systèmes d'exploitation
   - Unix
   - Solaris
   - Windows
   - Posix

Chapitre 3/4/5/6 - Parallélisme/Étude de cas

Introduction

- La plupart des langages utilisent
  - moniteurs
  - messages
  - E/S
  - procédures/fonctions
  - commandes gardées
  - objets

CSP : introduction

- C'est une notation (Occam, ccs, ...)
- E/S synchronisées (passage de messages synchrones)
- Désignation directe
- ? : lecture (réception)
- ! : écriture (envoi)

Exemple : tampon à N éléments

```
buf:: buffer:(0..9) Portion

      in, out:integer; in :=0; out :=0;

      *[ in < out + 10; producer?buffer(in mod 10)
          -> in := in +1

       [] out < in; consumer ? more()
           -> consumer ! buffer(out mod 10)
              out := out +1;
        ]
```

Exemple : tampon à N éléments

- Pcs producteur →
  `buf!p`
- Pcs consommateur →
  `buf!more(); buf?p;`

Syntaxe pour la concurrence

- Concurrence

  `[P1 | P2 | ....| Pn]`

- Sélection

  `[C1 [] C2 [] ....[] Cn]`

  Chaque `Ci` est une commande gardée du type

  garde $\rightarrow$ *liste de commandes*

- Itération

  `*[C1 [] C2 [] ....[] Cn]`

Exemple : appel de procédure

```
[ x:: *[
         y!entree(parametres...);
         y?retour(parametres...);
         ...
       ]
 |
  y:: *[
         x?entree(parametres...)
         x!retour(parametres...)
         ...
       ]
]
```

Exemple : factoriel

```
[ fact(i:1..maxproc)::
    *[ n: integer
       fact(i-1)?n -> [ n=0 -> fact(i-1!1;
                       [] n>0 -> fact(i+1)! n-1;

                                 temp: integer
                                 fact(i+1) ? temp;
                                 fact(i-1) ! n*temp;
                       ]
     ]
  | fact(0) :: user
]
```

Exemple : sémaphore

```
sem:: sem:integer; sem:=0;

   *[ (i:1..maxproc) USER(i)? v() -> sem := sem+1;

     [](i:1..maxproc) sem>0; USER(i)?p() -> sem:=sem-1;
    ]
```

Ada

- Développé par DOD (1970)
- Pour les applications embarquées
- Supporte la concurrence avec mémoire commune et passage de messages.

Syntaxe

- 5 unités de compilation :
  - les procédures,
  - les packages (body et specification) et
  - les task (body et specification)

Syntaxe : package

- Concept de base est le ≪package≫
- Sert à implanter des déclarations de types patagés, de types abstraits et des bibliothèques.
- Deux parties : ≪package specification≫ et ≪package body≫

Syntaxe : package

Package specification	**Package body**

package *name* **is**
Déclaration des procédures visibles (entêtes), des types, des donstantes et des variables
 	
end;
	package body *name* **is**
	Déclaration des procédures, des constantes, des variables et des corps complets des procédures.
	begin
	Initialisation
	[**exception**
	Gestionnaire des exception
	]
	end;

Syntaxe : task

- Ce sont les processus de ADA
- Deux parties : ≪task specification≫ et ≪task body≫
- La spécification sert à exporter les ≪entrées≫ (task entries) qui sont similaires aux procédures
- Elles peuvent être déclarées dans des procédures, des packages, des task ou des blocs de programmes.

Syntaxe : task

- Les tâches sont démarrées implicitement aussitôt qu'une instance de la tâche est connu.
- Les tâches peuvent accéder les variables globales
- Unifie le concept de moniteur et processus (Rendez-vous)

Syntaxe : package

Task specification	**Task body**

task *name* **is**
entry *spécifications*
end;
	task body *name* **is**
	Déclaration des procédures.
	begin
	énoncés de la tâche
	[**exception**
	Pilote des exceptions]
	end;

Syntaxe : entry et accept

- Un énoncé entry sert à la spécification
- Il a la même structure qu'une entête de procédure
- À chaque énoncé entry correspond, dans le corps de la tâche, un énoncé accept

  `accept` *entryname* (paramètres formels) `do`

  énoncés formant le corps de cette entrée

  `end` *entryname*

Syntaxe : entry et accept

- Soit une entrée **E** dans une tâche **T1**
- Un autre processus peut l'utiliser grâce à un appel
  T1.E(arguments)
- Un énoncé accept s'exécute lorsque le traitement normal de **T1** le rejoint et qu'une autre tâche a fait un appel (rendez-vous)
- Le code de l'énoncé accept s'exécute en parallèle avec le code de l'appelant.

Syntaxe : entry et accept

```
task tampon is
  entry place(x : in message)
  entry remove(x : out message)
end;
```

Syntaxe : entry et accept

```
task body tampon is
  buffer : message;
begin
  loop
    accept place(x: in message) do
      buffer := x;
    end place;
    accept remove(x : out message) do
      x:= buffer;
    end remove;
  end loop;
end tampon;
```

Syntaxe : select

- Ada permet de répondre aux événements dans un ordre quelconque grâce à l'énoncé select
- syntaxe

  enonce-select : := att-select | appel-cond | appel-chrono

  att-select : := select alt_1 { or alt_i} [else {énoncés}] end select

  alt_i : := [when condition =>] alt-attente

  alt-attente : := alt-accept | alt-delai | alt-fin

  alt-accept : := accept {énoncés}

  alt-delai : := enonce-delai {énoncés}

  alt-fin : := terminate

Syntaxe :select

- Au moins une alternative ≪alt-accept≫ doit être présente dans une attente sélective
- Une alternative dans une attente sélective est dite ≪ouverte≫ si elle n'est pas préfixée d'une garde ou si la condition de la garde est vraie

Syntaxe : select

```
task tampon is
  entry place(x : in message)
  entry remove(x : out message)
end;
```

Syntaxe : select

```
task body tampon is
  N : constant := 10;
  tampon : array (0 ..N-1) of message;
  i,j : integer range 0..N-1 : = 0;
  count : integer range 0..N :=o;
```

Syntaxe : select

```
begin
  loop
    select
      when count < N =>
         accept place(x: in message) do
             buffer := x;
         end place;
         i:=(i+1) mod N;
         count := count + 1;
    or
      when count > 0 =>
         accept remove(x : out message) do
            x:= buffer;
         end remove;
         j:=j+1 mod N;
         count := count -1;
    end select
  end loop;
end tampon;
```

Syntaxe : select

- Appel conditionnel permet à un appelant d'annuler un appel si un rendez-vous est impossible immédiatement
- syntaxe

select
énoncé d'appel
or
énoncés
end select

Syntaxe : select

- Appel chronométré est une variation d'un appel conditionnel
- syntaxe

select

énoncé d'appel

or

alternative de délai

end select

Syntaxe : select

- Exemple : Un appel à une pile est annulé s'il n'y a pas rendez-vous après 10 secondes

```
select
        pile.empile(para...)
or
        delay 10.0
end select
```

Syntaxe : select

```
task serveur is
  entry signal(para...)
  entry enable
  entry disable
end;
```

Syntaxe : select

```
task body serveur is
  type inhibit is (on, off)
  service : inhibit := off;
begin
  loop
    select
      when service = on =>
         accept signal (para...) do
              ...
         end signal;
    or
      accept disable do
            service := off;
      end disable;
    or
      delay 2; print ("pas de signal");
    end select
  end loop;
end tampon;
```

Syntaxe : select

```
-- COMPILE: ada phl.a -M phl -o phl
-- PURPOSE: tasking test and demonstration
-- DESCRIPTION: The dining philosophers problem.
-- ..............................................--
with text_io;
procedure phl is
   print_task_starts : constant boolean := true;
   term_type:      character := '1';
   subtype seat is integer range 0..4;

   task type output is
      entry put_cursor ( s : seat; at_table, eating : boolean);
      entry put_line(s: in string);
      entry put(s: in string);
      entry   term_type_entry(x: in character);
      entry   clear_screen;
   end output;

   o: output;
```

Syntaxe : select

```
task type dining_room is
        entry allocate_seat (s : out seat);
        entry enter;
        entry leave;
end dining_room;

dr:  dining_room;

task type fork is
        entry pick_up;
        entry put_down;
end fork;

cutlery : array (0..4) of fork;

task type philosopher;

school : array (0..4) of philosopher;

task rand_delay is
        entry rand;
end rand_delay;
```

Syntaxe : select

```
task body output is
  term_type: character;   -- '1' for vt100, '2' for f100
  clear_sc : constant string := ascii.esc & '*' ;
  use text_io;
  pnum : character;
  r_s : seat;
  r_at_table, r_eating : boolean;

  type xy_position is
          record
                x : integer range 0 .. 79;
                y : integer range 0 .. 23;
          end record;

  eating_coords : array (seat) of xy_position :=
  ( (28, 6), (25,12), (36,15), (46,12), (44, 6) );

  thinking_coords : array(seat) of xy_position :=
  ( (20,21), (30,21), (40,21), (50,21), (60,21) );
```

Syntaxe : select

```
procedure put( item: in string ; pos: in xy_position ) is
     xp : integer := pos.x; yp : integer := pos.y;
     elp : integer;
     type as is access string;
     s : as := new string'(item);
     xs : as := new string'(integer'image(xp));
     ys : as := new string'(integer'image(yp));
  begin
    if yp /= 23 then  elp := 79;
                else     elp := 78;
    end if; -- can't write 79,23
    if ( xp + s'length > elp ) then
      s := new string'(s( s'first .. s'first + elp - xp ));
    end if;
    -- position and write string
    if term_type = '1' then
        text_io.put( ascii.esc & "[" & ys(2..ys'last) & ";"
                & xs(2..xs'last) & "H" & s.all );
    else
       text_io.put( ascii.esc & "=" & character'val(32 + yp)
                & character'val(32 + xp) & s.all );
    end if;
end put;
```

Syntaxe : select

```
begin
  accept term_type_entry(x: in character) do
         term_type := x;
  end term_type_entry;
  loop
    select
      accept put_cursor(s:seat; at_table, eating:boolean) do
         r_s := s;   r_at_table := at_table;
         r_eating := eating;

         pnum := character'val(r_s + 1 + character'pos('0'));
         if r_at_table then
           put( "  ", thinking_coords(r_s) );
             if r_eating then
                  put("P" & pnum & "E", eating_coords(r_s));
             else
                  put("P" & pnum & " ", eating_coords(r_s));
             end if;
          else
                put( "   ", eating_coords(r_s) );
                put( "P" & pnum, thinking_coords(r_s) );
          end if;
      end put_cursor;
```

Syntaxe : select

```
      or
        accept put(s : in string) do
                     text_io.put(s);
        end put;
      or
        accept put_line(s : in string) do
                      text_io.put_line(s);
        end put_line;
      or
        accept clear_screen do
          if term_type = '1' then
             text_io.put(ascii.esc & "[2J" & ascii.esc & "[H");
          else
              text_io.put(clear_sc);
          end if;
          delay 0.1;
        end clear_screen;
    end select;
  end loop;
end output;
```

Syntaxe : select

```
task body dining_room is
    seats_filled : integer range 0..5 := 0;
    seat_allocation : seat := 0;
begin
   if print_task_starts then
           o.put_line("dining room starting");
   end if;
   o.clear_screen;
   o.put_line("                        non_stop eating and thinki
   o.put_line("   ");
   o.put_line("           ***");
   o.put_line("         *******");
   o.put_line("       **********");
   o.put_line("     ******   ******");
   o.put_line("    ****** @@@ ******");
   o.put_line("   *****  @@@@@ *****");
   o.put_line("    *****  @@@  *****");
   o.put_line("     *****     *****");
   o.put_line("      ************");
   o.put_line("       **********");
   o.put_line("   ");
   o.put_line("   ");
```

Syntaxe : select

```
loop
  select
    --allocate fixed seat numbers to each  philosophers
     accept allocate_seat (s : out seat) do
              s := seat_allocation;
              if seat_allocation < 4 then
                  seat_allocation := seat_allocation + 1;
              end if;
     end ;
  or when seats_filled < 5 =>
          accept enter do
                seats_filled := seats_filled + 1;
          end;
  or accept leave do
           seats_filled := seats_filled - 1;
      end;
  end select;
end loop

end dining_room;
```

Syntaxe : select

```
task body fork is
begin
    if print_task_starts then
    o.put_line("fork starting");
    end if;
    loop
    accept pick_up;
    accept put_down;
    end loop;
end fork;

task body rand_delay is
random : duration := 0.4;
begin
     loop
  random := random + 0.05;
  if random > 0.7 then random := 0.4; end if;
  accept rand do
  delay random;
  end rand;
     end loop;
end rand_delay;
```

Syntaxe : select

```
task body philosopher is
        s : seat;
begin
   if print_task_starts then
          o.put_line("philosopher starting");
   end if;
   dr.allocate_seat(s); --obtain seat on joining institution;
   dr.enter;
   o.put_cursor(s, at_table => true, eating => false);
   rand_delay.rand;
```

Syntaxe : select

```
loop
   cutlery(s).pick_up; (premiere fourchette)
   --essayer d'obtenir la seconde)
   select cutlery((s+1) mod 5).pick_up;
       -- philosopher begins to eat
       o.put_cursor(s, at_table => true, eating => true);
       delay 1.0;

       cutlery((s+1) mod 5).put_down;
       cutlery(s).put_down;
       --leave the room to think
       dr.leave;
       o.put_cursor(s, at_table => false, eating => false)
       delay 1.2;

       -- enter dining room again
       dr.enter;
       o.put_cursor(s, at_table => true, eating => false);
       delay 0.9;
```

Syntaxe : select

```
      or
           -- let someone else try for 2 forks
           delay 0.9;
           cutlery(s).put_down;
           rand_delay.rand;
      end select;
   end loop;
end philosopher;
```

Syntaxe : select

```
begin
   text_io.put("is this a vt100 (1) or an f100 (2) -->");
   text_io.get(term_type);
   if term_type /= '1' and term_type /= '2' then
         text_io.put(" ... i'll assume you have a vt100");
   end if;
   o.term_type_entry(term_type);
end phl;
```

Le langage Java

- Chaque programme Java contient au moins un fil d'exécution
- Chaque programme Java contient d'autres fils pour : ramasse-miette, finalisation des objets, housekeeping, ...
- Selon les outils que l'on utilise, on peut ajouter encore d'autres fils : AWT, SWING, Servlet, RMI, ...
- Ces fils sont invisibles
- Java fournit deux techniques pour créer explicitement de nouveaux fils

Classe Thread

- Un programme peut créer des fils en créant des instances de la classe Thread ou d'une classe qui en hérite
- Cet objet doit obligatoirement posséder une méthode du nom de «run»

```
public class .... extends Thread {
  ...
  public void run() {...}
}
```

Classe Thread

- Démarrer l'exécution d'un fil le programme doit appeler la méthode start()
- Déclaration du fil

```
class monThread extends Thread {
   monThread() {...}
   public void run() { ...}
}
```

- Dans le programme

```
monThread p = new monThread();
p.start();
```

Classe Thread

- Un fil se termine lorsque la méthode run termine
- Autres méthodes de la classe Thread

```
Sleep()
isAlive()
yield()
join()
interrupt()
```

Exemple 1

```
public class TwoThreads {
  public static class Thread1 extends Thread {
      public void run() {
          System.out.println("A");
          System.out.println("B");
      }
  }
  public static class Thread2 extends Thread {
      public void run() {
          System.out.println("1");
          System.out.println("2");
      }
  }
  public static void main(String[] args) {
      new Thread1().start();
      new Thread2().start();
  }
}
```

Exemple 2

```
public class CounterThread extends Thread {
  Counter c;
  public static void main(String[] args) {
    Counter c = new Counter();
    CounterThread ct1 = new CounterThread(c);
    CounterThread ct2 = new CounterThread(c);
    ct1.start(); ct2.start();
    try{
      ct1.join();  ct2.join();
    } catch (InterruptedException e)
              {System.out.println("Error during join");}
    System.out.println("Var i= "+c.get()+", expected i= 20");
  }
  public CounterThread(Counter c) {
    this.c = c;
  }
  public void run() {
    c.count();
  }
}
```

Exemple 2 (suite)

```
class Counter {
  int i = 0;
  public void count() {
    for (int j=0; j<10; j++){
      i=inc(i);
      try{
        Thread.sleep((int) (Math.random()*10));
      } catch (InterruptedException e)
              {System.out.println("Error during sleep");}
    }
  }
  public int get(){ return i; }
  private int inc(int n){
    try{
      Thread.sleep((int) (Math.random()*5));
    } catch (InterruptedException e)
             {System.out.println("Error during sleep");}
    return n+1;
  }
}
```

Exemple 3

```
public class TenThreads {
  private static class WorkerThread extends Thread {
      int max = Integer.MIN_VALUE;
      int[] ourArray;
      public WorkerThread(int[] ourArray) {
            this.ourArray = ourArray;
      }
      // Find the maximum value in our piece of the array
      public void run() {
            for (int i = 0; i < ourArray.length; i++)
            max = Math.max(max, ourArray[i]);
      }
      public int getMax() {
                  return max;
      }
   }
```

Exemple 3

```
public static void main(String[] args) {
    WorkerThread[] threads = new WorkerThread[10];
    int[][] bigMatrix = getBigHairyMatrix();
    int max = Integer.MIN_VALUE;
    // Give each thread a slice of the matrix
    for (int i=0; i < 10; i++) {
         threads[i] = new WorkerThread(bigMatrix[i]);
         threads[i].start();
    }
    try {
          for (int i=0; i < 10; i++) {
              threads[i].join();
              max = Math.max(max, threads[i].getMax());
          }
   }
   catch (InterruptedException e) {
                // fall through          }
   System.out.println("Maximum value was " + max);
}
```

Exemple 4

```
public class CalculatePrimes extends Thread {
  public static final int MAX_PRIMES = 1000000;
  public static final int TEN_SECONDS = 10000;
  public volatile boolean finished = false;
  public void run() {
     int[] primes = new int[MAX_PRIMES];
     int count = 0;
     for (int i=2; count<MAX_PRIMES; i++) {
         if (finished) { break; }
         boolean prime = true;
         for (int j=0; j<count; j++) {
             if (i % primes[j] == 0) {
               prime = false;
               break;
             }
         }
         if (prime) {
             primes[count++] = i;
             System.out.println("Found prime: " + i);
         }
     }
  }
```

Exemple 4

```
  public static void main(String[] args) {
    CalculatePrimes calculator = new CalculatePrimes();
    calculator.start();
    try {
         Thread.sleep(TEN_SECONDS);
        }
    catch (InterruptedException e) {
                  // fall through
    }
    calculator.finished = true;
  }
}
```

Classe Thread

- Créer une instance de l'objet Thread ne le démarre pas
- Un objet existe avant son démarrage et après la fin de son exécution
- Il ne faut pas démarrer un fil dans le constructeur
- Le fil peut terminer normalement (fin de run), sur une exception ou erreur, sur un stop()

Interface Runnable

- Un programme peut déclarer une classe qui implante l'interface Runnable
- Cette interface déclare seulement une méthode «run»
- On passe ensuite une instance de cette classe au constructeur de Thread

Interface Runnable

```
class monThread2 implements Runnable
{
  monThread2() { ...}
  public void run()
  { ... }

}

monThread2 p = new monThread2
Thread t = Thread(p)
t.start();
```

Synchronisation et communication

- Par partage de variables
- Deux outils de synchronisation
  - volatile
  - synchronized (implante le concept de moniteur)

Synchronisation : synchronized

- Il y a un verrou par objet
- Peut spécifier un objet particulier verrouiller

```
synchronized (IObjet)
{
    code....
}
```

Synchronisation : attente conditionnelle

- Il y a une variable condition par objet
- Opérations
  - wait()
  - notify (signal&continue)
  - notifyAll()
- Pour avoir plusieurs variables condition, on peut créer des nouveaux objets

Exemple 5

```
public class SyncExample {
     private static Object lockObject = new Object();
     private static int x;
     private static int y;
     private static class Thread1 extends Thread {
        public void run() {
             synchronized (lockObject) {
                 x = y = 0;
                 System.out.println(x);
             }
     }}
     private static class Thread2 extends Thread {
          public void run() {
             synchronized (lockObject) {
                   x = y = 1;
                   System.out.println(y);
             }
     }}
     public static void main(String[] args) {
             new Thread1().run();
             new Thread2().run();
     }
}
```

Exemple 6

```
public class ProducerConsumerTest
{
    public static void main(String[] args)
    {
        CubbyHole c = new CubbyHole();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);

        p1.start();
        c1.start();
    }
}
```

Exemple 6

```
public class Producer extends Thread
{
    private CubbyHole cubbyhole; // shared data
    private int number;
    public Producer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }
    public void run()   {
       for (int i = 0; i < 10; i++) {
           cubbyhole.put(i); // Accesses shared data
           System.out.println("Producer #" + this.number
                              + " put: " + i);
           try   {
              sleep((int)(Math.random() * 100));
           } catch (InterruptedException e) { }
       }
   }
}
```

Exemple 6

```
public class Consumer extends Thread
{
    private CubbyHole cubbyhole;  // shared data
    private int number;
    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }
    public void run()  {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get();
            System.out.println("Consumer #" + this.number
                                + " got: " + value);
        }
    }
}
```

Exemple 6

```
public class CubbyHole {
    private int contents; // shared data
    private boolean available = false;
    public synchronized int get()     {
        while (available == false) {
            try   {   wait(); }
            catch (InterruptedException e) { }
        }
        available = false;
        notifyAll();
        return contents;
    }
    public synchronized void put(int value) {
        while (available == true) {
            try { wait(); }
            catch (InterruptedException e) { }
        }
        contents = value;
        available = true;
        notifyAll();
    }
}
```

Exemple 7

```
import java.io.*;
import java.util.*;
public class Test {
   private Set lockSet = new HashSet();
   public void lock(int recNo) {
        Integer key = new Integer(recNo);
        synchronized(lockSet) {
            while(lockSet.contains(key)) {
               try{ lockSet.wait();  }
               catch(InterruptedException ex) {}
            }
            lockSet.add(key);
            System.out.println("locked "+recNo);
        }
   }
   public void unlock(int recNo) {
      try {Thread.sleep(500);}
      catch(InterruptedException ex) {}
      Integer key = new Integer(recNo);
      synchronized(lockSet) {
             lockSet.remove(key);
             System.out.println("unlocked "+recNo);
                 lockSet.notifyAll();
         }
  }
```

Exemple 7

```
public static void main(String[] args) throws Exception
{
      Test test = new Test();
      test.testLocking();
}
private void testLocking() {
   lockIt(5); lockIt(6); lockIt(6); lockIt(6);
   lockIt(6); lockIt(6); lockIt(6); lockIt(6);
   lockIt(6); lockIt(5);
   unlockIt(5);
}
```

Exemple 7

```
 private void lockIt(final int x) {
     new Thread(new Runnable() {
           public void run() {
              try {
                   lock(x);
              }
              catch(Exception e) {e.printStackTrace();}
          }
     }).start();
  }
  private void unlockIt(final int x) {
     new Thread(new Runnable() {
          public void run() {
              try {
                   unlock(x);
              }
              catch(Exception e) {e.printStackTrace();}
         }
    }).start();
  }
}
```

Synchronisation

- Il existe d'autres implantations pour la synchronisation
  - abstractMonitor
  - concurrent.util (verrou, mutex, sémaphore, queues, ...)

Communication : RMI

- Pour la relation client/serveur
- Le serveur :
  - crée des objets éloignés
  - met des références disponibles vers ces objets
  - attend des appels
- Le client :
  - obtient des références
  - appelle les fonctions
- Avec RMI les références sont mises disponibles dans le registre RMI

Communication : RMI

- Pour le cas particulier de RMI
- Le serveur crée des interfaces pour soumettre ses tâches
- Chaque interface contient une seule méthode
- Cette interface permet aux clients de faire des appels

Communication : RMI

```
//  Definition de l'interface

import java.rmi.*;

public interface SampleServer extends Remote
{
  public int sum(int a,int b) throws RemoteException;
}
```

Communication : RMI

- Pour le cas particulier de RMI
- Une fois l'interface créé, le serveur crée l'objet qui implante l'interface
- Il enregistre l'interface dans le serveur de nom (registre) grâce à java.rmi.naming

Communication : RMI

```
//////////////////////////////////////
// Definition de l'objet du coté serveur
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class SampleServerImpl extends UnicastRemoteObject
                                implements SampleServer      {
  SampleServerImpl() throws RemoteException {
     super();
  }
  public int sum(int a,int b) throws RemoteException  {
     return a + b;
  }
```

Communication : RMI

```
  public static void main(String args[])  {
    try {
        System.setSecurityManager(new RMISecurityManager());
        //create a local instance of the object
        SampleServerImpl Server = new SampleServerImpl();
        //put the local instance in the registry
        Naming.rebind("SAMPLE-SERVER" , Server);

        System.out.println("Server waiting.....");
    }
    catch (java.net.MalformedURLException me)      {
         System.out.println("Malformed URL: " + me.toString());
    }
    catch (RemoteException re)     {
         System.out.println("Remote exception: " + re.toString());
      }
  }
}
```

Communication : RMI

- Le client recherche d'abord le nom du serveur dans le registre
- Il appelle la méthode

Communication : RMI

```
import java.rmi.*;
import java.rmi.server.*;
public class SampleClient  {
   public static void main(String[]  args)  {
      System.setSecurityManager(new RMISecurityManager());
      //get the remote object from the registry
      try{
          System.out.println("Security Manager loaded");
          String url = "//localhost/SAMPLE-SERVER";
          SampleServer remoteObject = (SampleServer)Naming.lookup(url);
          System.out.println("Got remote object");
          System.out.println(" 1 + 2 = " +
               remoteObject.sum(1,2) );
        }
      catch (RemoteException exc) {
          System.out.println("Error in lookup: " + exc.toString());
        }
      catch (java.net.MalformedURLException exc)   {
          System.out.println("Malformed URL: " + exc.toString());
        }
      catch (java.rmi.NotBoundException exc)     {
          System.out.println("NotBound: " + exc.toString());
        }
   }
}
```

Communication : RMI

- On compile le code java
- On génère le stub et le squelette du serveur
- On démarre le registre RMI
- On démarre le serveur
- On démarre le client

Le langage Python

- Les fils de Python utilisent les fils de niveau noyau du système sous-jacent
- Le GIL de Python (Global Interpetor Lock) contrôle les changements de contexte
- Ne profite donc pas des multi-processeurs
- L'interpréteur Python est multi-fils
- On implante un programme multi-fils en Python grâce aux modules `Thread` (thread.py) et `Threading` (Threading.py).

Module Threading

- C'est le plus évolué
- Il faut créer une sous-classe de la classe Thread et définir une méthode ≪run≫
- Pour exécuter un fil, on crée une ou plusieurs instances de la classe et on appelle la méthode start()
- Il y a un fil par instance

Module Threading

```
import threading
...


class test ( threading.Thread ):

   def _init_(self)
   ...
   def run ( self ):...

for i in range ( 10 ):
   test().start()
```

Exemple 1

```
import threading

theVar = 1

class MyThread ( threading.Thread ):

   def run ( self ):

      global theVar
      print 'This is thread '+ str ( theVar )+' speaking.'
      print 'Hello and good bye.'
      theVar = theVar + 1

for x in xrange ( 20 ):
   MyThread().start()
```

Exemple 2

```
#!/usr/bin/env python

import threading

class MyThread ( threading.Thread ):

   def run ( self ):

      print 'You called my start method, yeah.'
      print 'Were you expecting something amazing?'

MyThread().start()
```

Exemple 3

```
import time
from threading import Thread

class MyThread(Thread):

    def __init__(self,bignum):
        Thread.__init__(self)
        self.bignum=bignum

    def run(self):
        for l in range(10):
            for k in range(self.bignum):
                res=0
                for i in range(self.bignum):
                    res+=1
```

Exemple 3

```
def test():
    bignum=1000
    thr1=MyThread(bignum)
    thr1.start()
    thr1.join()

if __name__=="__main__":
    test()
```

Module Thread

- C'est le plus primitif
- On fait : `import Thread`
- On définit une fonction
- On exécute la fonction en faisant

```
Thread.start.new.thread(nom_fct(), ...)
```

Exemple 4

```
#!/usr/bin/env python

import time
import thread

def myfunction(string,sleeptime,*args):
    while 1:

        print string
        time.sleep(sleeptime) #sleep for a time.

if __name__=="__main__":

    thread.start_new_thread(myfunction,("Thread No:1",2))

    while 1:pass
```

Synchronisation

- Fournit
  - Verrous
  - Moniteurs (condition)
  - Sémaphores
  - Événements

Synchronisation : verrous

- Fournis par la classe Threading
- On peut créer, acquérir et relâcher un verrou :
  - lock = Threading.Lock
  - lock.Acquire() ou lock.Acquire(false)
  - lock.release()
  - lock.Locked()
- Utiliser souvent avec «with» : `with lock : ....`

Synchronisation : verrous

- Il y a aussi des verrous réentrant : RLock
- lock = Threading.RLock

```
lock = Threading.Lock()
lock.acquire()
lock.acquire()   -> bloqué

lock = Threading.RLock()
lock.acquire()
lock.acquire()   -> non bloqué
```

- les verrous sont aussi fournis pas la classe Thread : Thread.allocate.Lock()

Synchronisation : sémaphore

- Fournis par la classe Threading
- Il y a deux types de sémaphores (ordinaire et bounded)

```
    sema = Threading.Bounded.Semaphore()
ou  sema = Threadind.Semaphore()
    sema.acquire()
    sema.release()
```

- On peut initialiser un sémaphore
  `sema = Threading.Bounded.semaphore(10)`
- Bounded limite le nombre de release (ne doit pas dépasser le nombre de acquire)

Synchronisation : événements

- ev = Threading.Event()
- ev.wait() — attend qu'il soit positionné
- ev.set()
- ev.clear()

Synchronisation : moniteur (condition)

- Fournis par Threating
- création : cond = Threading.Condition()
- Opérations :
  - cond.acquire() — entrée dans moniteur
  - cond.release() — sortie du moniteur
  - cond.wait() ou cond.wait(délai en secondes)
  - cond.notify() (signal&continue)
  - cond.notifyAll()

Synchronisation : moniteur (condition)

- On peut associer une condition à un verrou
- lock = Threading.RLock()
- cond1 = Threading.condition(lock)
- cond2 = Threading.condition(lock)

Exemple 5

```
#!/usr/bin/env python
import time
import thread

def myfunction(string,sleeptime,lock,*args):
    while 1:
        lock.acquire()
        print string," Sleeping Lock acquired ",sleeptime
        time.sleep(sleeptime)
        print string," Now releasing lock "
        lock.release()
        time.sleep(sleeptime) # why?

if __name__=="__main__":

  lock=thread.allocate_lock()
  thread.start_new_thread(myfunction,("Thread 1",2,lock))
  thread.start_new_thread(myfunction,("Thread 2",2,lock))

  while 1:pass
```

Exemple 6

```
class itemQ:

    def __init__(self):
        self.count=0

    def produce(self,num=1):
        self.count+=num

    def consume(self):
        if self.count: self.count-=1

    def isEmpty(self):
        return not self.count
```

Exemple 6

```
class Producer(Thread):

    def __init__(self,condition,itemq,sleeptime=1):
        Thread.__init__(self)
        self.cond=condition
        self.itemq=itemq
        self.sleeptime=sleeptime

    def run(self):
        cond=self.cond
        itemq=self.itemq
        while 1 :
            cond.acquire() #acquire the lock
            print currentThread(),"Produced One Item"
            itemq.produce()
            cond.notifyAll()
            cond.release()

            time.sleep(self.sleeptime)
```

Exemple 6

```
class Consumer(Thread):

    def __init__(self,condition,itemq,sleeptime=2):
        Thread.__init__(self)
        self.cond=condition
        self.itemq=itemq
        self.sleeptime=sleeptime

    def run(self):
        cond=self.cond
        itemq=self.itemq
        while 1:
            time.sleep(self.sleeptime)
            cond.acquire() #acquire the lock
            while itemq.isEmpty():
                cond.wait()
            itemq.consume()
            print currentThread(),"Consumed One Item"
            cond.release()
```

Exemple 6

```
if __name__=="__main__":

    q=itemQ()

    cond=Condition()

    pro=Producer(cond,q)
    cons1=Consumer(cond,q)
    cons2=Consumer(cond,q)

    pro.start()
    cons1.start()
    cons2.start()
```

Communication : Pyro

- Fontionnement similaire au RMI de Java
- Il utilise un serveur de nom qui enregistre le nom des objets
- Les clients doivent interroger le serveur de nom et appeler la méthode

Exemple 1

```
# tst.py
# These classes will be remotely accessed.
class testclass:
  def mul(s, arg1, arg2): return arg1*arg2
  def add(s, arg1, arg2): return arg1+arg2
  def sub(s, arg1, arg2): return arg1-arg2
  def div(s, arg1, arg2): return arg1/arg2
  def error(s):
        x=foo()
        x.crash()
class foo:
  def crash(s):
       s.crash2('going down...')
  def crash2(s, arg):
       # this statement will crash on purpose:
       x=arg/2
```

Exemple 1

```
## SERVEUR

import Pyro.core
import Pyro.naming
import tst
class testclass(Pyro.core.ObjBase, tst.testclass):
  def __init__(self):
     Pyro.core.ObjBase.__init__(self)

Pyro.core.initServer()
ns=Pyro.naming.NameServerLocator().getNS()
daemon=Pyro.core.Daemon()
daemon.useNameServer(ns)
uri=daemon.connect(testclass(),"simple")

print "Server is ready."
daemon.requestLoop()
```

Exemple 1

```
## CLIENT
import Pyro.util
import Pyro.core
Pyro.core.initClient()

test = Pyro.core.getProxyForURI("PYRONAME://simple")

print test.mul(111,9)
print test.add(100,222)
print test.sub(222,100)
print test.div(2.0,9.0)
print test.mul('.',10)
print test.add('String1','String2')

print '*** invoking server method that crashes ***'
print test.error()
```

Exemple 2

Server:

```
import Pyro.core
import Pyro.naming

class JokeGen(Pyro.core.ObjBase):
        def __init__(self):
                Pyro.core.ObjBase.__init__(self)
        def joke(self, name):
                return "Sorry "+name+", I don't know any jokes."

Pyro.core.initServer()
ns=Pyro.naming.NameServerLocator().getNS()
daemon=Pyro.core.Daemon()
daemon.useNameServer(ns)
uri=daemon.connect(JokeGen(),"jokegen")
daemon.requestLoop()
```

Exemple 2

```
Client:

import Pyro.core

# finds object if you're running the Name Server.
jokes = Pyro.core.getProxyForURI("PYRONAME://jokegen")

print jokes.joke("Irmen")
```

Le langage C#

On peut créer des fils grâce à la classe «Thread» qui se trouve dans l'espace de nom «System.Threading»

Création de fil

- Importation de la classe Thread
```
using System;
using System.Threading;
```
- Déclaration d'un objet de type Thread et création d'une instance
```
Thread monFil;
monFil=new Thread(new ThreadStart(nomFonction));
```
- Démarrage du fil
```
monFil.start();
```

Création d'un fil

- `ThreadStart` est un concept spécifique à .NET appelé un délégué. C'est similaire à un pointeur de fonction
- On peut aussi écrire

```
System.Threading.Thread monfil;
```

- Ou bien

```
Thread monFil = new Thread(nomFonction);
monFil.start();
```

Attributs d'un fil

- Thread.currentThread.IsAlive (retourne un booléen)
- Thread.currentThread.name = "nom" ;
- monFil.IsBackground = true (arrête avec la fin de main)
- ...

Opérations sur un fil

- monFil.sleep(x ms)
- monFil.abort()
- monFil.join()
- ...

Exemple 1

```
using System;
using System.Threading;
class ThreadedApp
{
   public static void Main()
   {
      Thread myThread;
      myThread = new Thread(new ThreadStart(ThreadLoop));
      // Lancement du thread
      myThread.Start();
   }
   public static void ThreadLoop()
   {
      while (Thread.CurrentThread.IsAlive)
      {
         Thread.Sleep(500);
         Console.WriteLine("Je travaille...");
      }
  }
}
```

Exemple 2

```
using System;
using System.Threading;

class ThreadTest {
  static void Main() {
    Thread t = new Thread (new ThreadStart (Go));
    t.Start(); // Run Go() on the new thread.
    Go();      // Simultaneously run Go() in main.
  }
  static void Go() { Console.WriteLine ("hello!"); }
}
```

Exemple 3

```
using System;
using System.Threading;

class ThreadTest {

  static void Main() {
    Thread t = new Thread (WriteY);
    t.Start();                    // WriteY sur nouveau fil
    // Ecrire 'x' pour toujours
    while (true) Console.Write ("x");
  }
  static void WriteY() {

    while (true) Console.Write ("y");// Ecrire 'y'

  }
}
```

Exemple 4

```
using System;
using System.Threading;

static void Main() {
  new Thread (Go).Start(); //  Go() sur fil
  Go();                    //  Go() sur main
}
static void Go() {
  // Declare and use a local variable - 'cycles'
  for(int cycles=0; cycles<5; cycles++)
          Console.Write('?');
}
```

Exemple 5

```
using System;
using System.Threading;

public class A1
{
    // méthode appelée au démarrage du fil
    public void Beta()
    {
        while (true)
        {
          Console.WriteLine("A1.Beta execute sur son fil.");
        }
    }
};
```

Exemple 5 (suite)

```
public class Simple
{
  public static int Main()
  {
    Console.WriteLine("Thread Start/Stop/Join Sample");

    A1 oA1 = new A1();
    Thread oThread=new Thread(new ThreadStart(oA1.Beta));
    oThread.Start();
    while (!oThread.IsAlive);
    Thread.Sleep(1);
    oThread.Abort();
    oThread.Join();

    Console.WriteLine();
    Console.WriteLine("A1.Beta has finished");
    return 0;
  }
}
```

Exemple

```
using System;
using System.Threading;
static void Main()
{

  // Pas nécessaire d'utiliser explicitement ThreadStart
  Thread t = new Thread (Go);

  t.Start();
  ...
}

static void Go() { ... }
```

Synchonisation

- Tous les outils de synchonisation sont fournis par le framework .NET
- Les outils de synchronisation sont :
  - les moniteurs
  - les mutex
  - les sémaphores
  - les événements
  - les verrous lecteurs/écrivains

Les moniteurs

- Implantés par la classe monitor
  - monitor.Enter()
  - monitor.Exit()
  - monitor.Wait()
  - monitor.Pulse() ou monitor.PulseAll()
- Dans C# on utilise plutôt l'énoncé "lock"

```
lock (instance objet) énoncés
```

Les moniteurs

- Chaque objet implante une variable condition
- Les variables condition sont de type "signal&continue" (tout comme Java)

Exemple 1

```
private void DebiterCompte(int Montant)
{
  // Le code dans le bloc suivant sera protégé
  lock (this)
  {
    Console.WriteLine("Solde avant : " + Solde);
    Console.WriteLine("Montant à débiter : " + Montant);
    // Code critique
    Solde = Solde - Montant;
    Console.WriteLine("Solde après : " + Solde);
  }
}
```

Exemple 2

```
public static KV GetFromList()
{
   KV res;
   lock (typeof(KV))
   {
      while (head == null) Monitor.Wait(typeof(KV));
      res = head; head = res.next;
      res.next = null; // for cleanliness
   }
   return res;
}
public void AddToList()
{
   lock (typeof(KV))
   {
      /* We are assuming this.next == null */
      this.next = head; head = this;
      Monitor.Pulse(typeof(KV));
   }
}
```

Exemple 3

```
using System;
using System.Threading;
using System.Collections.Generic;

class ProducerConsumerQueue : IDisposable {
  EventWaitHandle wh = new AutoResetEvent (false);
  Thread worker;
  object locker = new object();
  Queue<string> tasks = new Queue<string>();

  public ProducerConsumerQueue() {
    worker = new Thread (Work);
    worker.Start();
  }
  public void EnqueueTask (string task) {
    lock (locker) tasks.Enqueue (task);
    wh.Set();
  }
```

Exemple 3 (suite)

```
  public void Dispose() {
    EnqueueTask (null); // Signal the consumer to exit.
    worker.Join();
    wh.Close();
  }
  void Work() {
    while (true) {
      string task = null;
      lock (locker)
        if (tasks.Count > 0) {
          task = tasks.Dequeue();
          if (task == null) return;
        }
      if (task != null) {
        Console.WriteLine ("Performing task: " + task);
        Thread.Sleep (1000);  // simulate work...
      }
      else
        wh.WaitOne(); //No more tasks - wait for signal
    }
  }
}
```

Exemple 4

```
using System;
using System.Threading;
public class MonitorSample  {
  public static void Main(String[] args) {
    int result = 0;
    Cell cell = new Cell( );
    CellProd prod = new CellProd(cell, 20);
    CellCons cons = new CellCons(cell, 20);
    Thread producer=new Thread(new ThreadStart(prod.ThreadRun));
    Thread consumer=new Thread(new ThreadStart(cons.ThreadRun));
    try {
       producer.Start( );  consumer.Start( );
       producer.Join( );   consumer.Join( );
    }
    catch (ThreadStateException e) {
       Console.WriteLine(e);
       result = 1;
    }
    catch (ThreadInterruptedException e) {
       Console.WriteLine(e);
       result = 1;
    }
    Environment.ExitCode = result;
  }
}
```

Exemple 4 (suite)

```
public class CellProd
{
   Cell cell;
   int quantity = 1;

   public CellProd(Cell box, int request)
   {
      cell = box;
      quantity = request;
   }
   public void ThreadRun( )
   {
      for(int looper=1; looper<=quantity; looper++)
         cell.WriteToCell(looper);  // "producing"
   }
}
```

Exemple 4 (suite)

```
public class CellCons
{
   Cell cell;
   int quantity = 1;

   public CellCons(Cell box, int request)
   {
      cell = box;
      quantity = request;
   }
   public void ThreadRun( )
   {
      int valReturned;
      for(int looper=1; looper<=quantity; looper++)
         valReturned=cell.ReadFromCell( );
   }
}
```

Exemple 4 (suite)

```
public class Cell {
  int cellContents;          // Cell contents
  bool readerFlag = false;   // State flag
  public int ReadFromCell( ) {
    lock(this)  {
       if (!readerFlag) {
          try  {
             Monitor.Wait(this);
          }
          catch (SynchronizationLockException e) {
             Console.WriteLine(e);
          }
          catch (ThreadInterruptedException e) {
              Console.WriteLine(e);
          }
       }
       Console.WriteLine("Consume: {0}",cellContents);
       readerFlag = false;
       Monitor.Pulse(this);
    }   // Sortie du moniteur
     return cellContents;
}
```

Exemple 4 (suite)

```
public void WriteToCell(int n)
{
   lock(this)  {
      if (readerFlag)   {
         try  {
            Monitor.Wait(this);
         }
         catch (SynchronizationLockException e) {
            Console.WriteLine(e);
         }
         catch (ThreadInterruptedException e) {
            Console.WriteLine(e);
         }
      }
      cellContents = n;
      Console.WriteLine("Produce: {0}",cellContents);
      readerFlag = true;
      Monitor.Pulse(this);
   }   // Exit synchronization block
}
```

Mutex

- Les mutex sont dérivés des WaitHandle de .NET
- Font appel à CreateMutex de WIN32
- Fonctionne entre les processus et entre les fils (Process wide lock)
- Le nom d'un mutex permet à plusieurs applications d'utiliser le même mutex

Mutex

- Création

```
Static Mutex monMutex = new Mutex(false,"nomMutex");
```

- Utilisation

```
monMutex.WaitOne();
monMutex.ReleaseMutex();
monMutex.Close();
monMutex.WaitAny();
monMutex.WaitAll();
```

Exemple

```
class OneAtATimePlease {
  static Mutex mutex = new Mutex (false, "UnALaFois");

  static void Main() {
    if (!mutex.WaitOne(TimeSpan.FromSeconds (5),false)) {
      Console.WriteLine("Another instance running. Bye!");
      return;
    }
    try {
      Console.WriteLine ("Running - press Enter to exit");
      Console.ReadLine();
    }
    finally { mutex.ReleaseMutex(); }
  }
}
```

Semaphores

- Ils sont dérivés des WaitHandle de .NET
- Peuvent être locaux ou globaux
- Fonctionne entre les processus et entre les fils (Process wide lock)
- Contrairement aux autres outils (mutex et moniteur) une sémaphore peut etre acquis par un fil et libéré par un autre

Sémaphore

- Création

```
Static Semaphore monSem = new Semaphore(3,3);
```

- Utilisation

```
monSem.WaitOne();
monSem.Release();
monSem.WaitAny();
monSem.WaitAll();
```

Exemple

```
class SemaphoreTest {
  // valeur init.=3; Capacite max=3
  static Semaphore s = new Semaphore (3, 3);
  static void Main()
  {
    for (int i = 0; i < 10; i++) new Thread (Go).Start();
  }
  static void Go() {
    while (true) {
      s.WaitOne();
      Thread.Sleep (100); //Seulement 3 fils simultanement
      s.Release();
    }
  }
}
```

Evénements

- Ils sont dérivés des WaitHandle de .NET
- Ils ont deux états : signalés ou non
- Deux types : AutoResetEvent et ManualResetEvent

Evénements : auto Reset

- Création

```
EventWaitHandle E = new AutoResetEvent(false);
```

- Utilisation

```
E.set();
E.WaitOne;
E.Reset;
```

- Une fois un événement signalé il le reste jusqu'à ce qu'un WaitOne soit fait
- Le WaitOne fait un Reset automatique sur l'événement

Evénements : Manual Reset

- Identique sauf que le WaitOne ne fait pas un Reset
- le Reset doit être fait explicitement

Exemple

```
using System;
using System.Threading;
public class MutexSample {
   static Mutex gM1;
   static Mutex gM2;
   const int ITERS = 100;
   static AutoResetEvent Event1=new AutoResetEvent(false);
   static AutoResetEvent Event2=new AutoResetEvent(false);
   static AutoResetEvent Event3=new AutoResetEvent(false);
   static AutoResetEvent Event4=new AutoResetEvent(false);

   public static void Main(String[] args)  {
      Console.WriteLine("Mutex Sample ...");
      gM1 = new Mutex(true,"MyMutex");
      gM2 = new Mutex(true);
      Console.WriteLine(" - Main Owns gM1 and gM2");

      AutoResetEvent[] evs = new AutoResetEvent[4];
      evs[0] = Event1;     // Event for t1
      evs[1] = Event2;     // Event for t2
      evs[2] = Event3;     // Event for t3
      evs[3] = Event4;     // Event for t4
```

Exemple (suite)

```
        MutexSample tm = new MutexSample( );
        Thread t1 = new Thread(new ThreadStart(tm.t1Start));
        Thread t2 = new Thread(new ThreadStart(tm.t2Start));
        Thread t3 = new Thread(new ThreadStart(tm.t3Start));
        Thread t4 = new Thread(new ThreadStart(tm.t4Start));
        t1.Start( ); // Does WaitAll(Mutex[] of gM1 and gM2)
        t2.Start( ); // Does WaitOne(Mutex gM1)
        t3.Start( ); // Does WaitAny(Mutex[] of gM1 and gM2)
        t4.Start( ); // Does WaitOne(Mutex gM2)

        Thread.Sleep(2000);
        Console.WriteLine(" - Main releases gM1");
        gM1.ReleaseMutex( ); //t2 and t3 will end and signal

        Thread.Sleep(1000);
        Console.WriteLine(" - Main releases gM2");
        gM2.ReleaseMutex( ); //t1 and t4 will end and signal

        // Waiting until all four threads signal done.
        WaitHandle.WaitAll(evs);
        Console.WriteLine("... Mutex Sample");
    }
```

Exemple (suite)

```
public void t1Start( )
{
 Console.WriteLine("t1Start started");
 Mutex[] gMs = new Mutex[2];
 gMs[0] = gM1;
 gMs[1] = gM2;
 Mutex.WaitAll(gMs); //Waits both gM1 and gM2 released
 Thread.Sleep(2000);
 Console.WriteLine("t1Start WaitAll() satisfied");
 Event1.Set( ); // AutoResetEvent.Set()  method is done
}

public void t2Start( )
{
 Console.WriteLine("t2Start started,  gM1.WaitOne( )");
 gM1.WaitOne( );  // Waits until Mutex gM1 is released
 Console.WriteLine("t2Start, WaitOne( ) satisfied");
 Event2.Set( );
}
```

Exemple (suite)

```
    public void t3Start( )
    {
      Console.WriteLine("t3Start started");
      Mutex[] gMs = new Mutex[2];
      gMs[0] = gM1;
      gMs[1] = gM2;
      Mutex.WaitAny(gMs);  // Waits either Mutex released
      Console.WriteLine("t3Start, Mutex.WaitAny(Mutex[])");
      Event3.Set( );
    }

    public void t4Start( )
    {
      Console.WriteLine("t4Start started,  gM2.WaitOne( )");
      gM2.WaitOne( );   // Waits until Mutex gM2 is released
      Console.WriteLine("t4Start, gM2.WaitOne( )");
      Event4.Set( );
    }
}
```

Verrous lecteurs/écrivains

- ReaderWriterLock.AcquireReaderLock
- ReaderWriterLock.AcquireWriterLock
- ReaderWriterLock.ReleaseLock
- ReaderWriterLock.UpgradeTowriterLock
- ReaderWriterLock.DowngradeFromWriterLock

Syntaxe : select

```
class Program {
  static ReaderWriterLock rw = new ReaderWriterLock ();
  static List <int> items = new List <int> ();
  static Random rand = new Random ();

  static void Main (string[] args) {
    new Thread (delegate(){while(true) AppendItem(); }).Start();
    new Thread (delegate(){while(true) RemoveItem(); }).Start();
    new Thread (delegate(){while(true) WriteTotal(); }).Start();
    new Thread (delegate(){while(true) WriteTotal(); }).Start();
  }
  static int GetRandNum(int max){lock(rand) return rand.Next(max);}
  static void WriteTotal() {
    rw.AcquireReaderLock (10000);
    int tot = 0; foreach (int i in items) tot += i;
    Console.WriteLine (tot);
    rw.ReleaseReaderLock();
  }
```

Exemple

```
 static void AppendItem () {
    rw.AcquireWriterLock (10000);
    items.Add (GetRandNum (1000));
    Thread.SpinWait (400);
    rw.ReleaseWriterLock();
  }

  static void RemoveItem () {
    rw.AcquireWriterLock (10000);
    if (items.Count > 0)
      items.RemoveAt (GetRandNum (items.Count));
    rw.ReleaseWriterLock();
  }
}
```

Introduction

- Fourni par la STL et Boost
- Accessible aussi par la bibliothèque Posix

Exemple 1 (STL)

```
#include <string>
#include <iostream>
#include <thread>

using namespace std;
void task1(string msg)
{  cout << "task1 says: " << msg;
}

int main()
{
    thread t1(task1, "Hello");

    t1.join();
}
```

Exemple 2 (STL)

```
#include <string>
#include <iostream>
#include <thread>
using namespace std;
void task1() {
    for (int i=0; i< 50; i++)   {
       cout << "11 : " << i << endl;
    }}
void task2() {
    for (int i=0; i< 50; i++)  {
        cout << "2222222222 : "<< i << endl;
    }}
int main (int argc, char ** argv) {
    thread thread_1 = thread(task1);
    thread thread_2 = thread(task2);
    // do other stuff
    thread_2.join(); thread_1.join();
    return 0;
}
```

Exemple 3 (STL)

```
#include <iostream>
#include <thread>
void foo()
{
    for (int i=0; i< 50; i++)   {
        std::cout << i << std::endl;
    }}
void bar(int x)
{
    for (int i=0; i< 50; i++)   {
        std::cout << x << std::endl;
    }}
int main()
{
  std::thread first (foo);
  std::thread second (bar,0);
  std::cout << "main, foo and bar now sont concurrents";
  // synchronize threads:
  first.join();     second.join();
  return 0;
}
```

Exemple 4 (STL)

```
#include <thread>
#include <iostream>
#include <vector>

void hello(){
  std::cout << "Allo du fil "
            << std::this_thread::get_id() << std::endl;
}

int main(){
  std::vector<std::thread> threads;

  for(int i = 0; i < 5; ++i){
    threads.push_back(std::thread(hello));   }

  for(auto& thread : threads){
    thread.join();  }

    return 0;
}
```

Exemple 5 (STL)

```
#include <iostream>
#include <thread>
#include <mutex>
#include <condition_variable>

std::mutex mtx;
std::condition_variable cv;
bool ready = false;

void print_id (int id) {
  std::unique_lock<std::mutex> lck(mtx);
  while (!ready) cv.wait(lck);
  // ...
  std::cout << "thread " << id << '\n';
}
void go() {
  std::unique_lock<std::mutex> lck(mtx);
  ready = true;
  cv.notify_all();
}
```

Exemple 5 (STL) (suite)

```
int main ()
{
  std::thread threads[10];
  // spawn 10 threads:
  for (int i=0; i<10; ++i)
    threads[i] = std::thread(print_id,i);
  std::cout << "10 threads ready to race...\n";
  go();
  for (auto& th : threads) th.join();
  return 0;
}
```

Exemple 6 (suite)

```
#include <iostream>        // std::cout
#include <thread>          // std::thread
static const int num_threads = 10;
void bar(int x)
{
   std::cout << x<< std::endl;
}

int main() {
    std::thread t[num_threads];

    for (int i = 0; i < num_threads; ++i) {
        t[i] = std::thread(bar, i);
    }
    std::cout << "Launched from the main\n";

    for (int i = 0; i < num_threads; ++i) {
        t[i].join();
    }
    return 0;
}
```

Exemple 7 (STL)

```
#include <string>
#include <iostream>
#include <thread>

using namespace std;

void task1(string msg)
{  cout << "task1 says: " << msg;
}

int main()
{
    thread t1(task1, "Hello");

    t1.join();
}
```

Exemple 8 (STL)

```
#include <iostream>
#include <thread>
#include <mutex>
#include <chrono>

using namespace std;
mutex g_lock;
void func()  {
    g_lock.lock();
    cout << "fil # " << this_thread::get_id() << endl;
    this_thread::sleep_for(chrono::seconds(rand() % 10));
    cout << "fin fil # " << this_thread::get_id() << endl;
    g_lock.unlock();
}
int main() {
    srand((unsigned int)time(0));
    std::thread t1(func);
    std::thread t2(func);
    std::thread t3(func);
    t1.join(); t2.join(); t3.join();
    return 0;
}
```

Exemple 9 (STL)

```
#include <condition_variable>
#include <mutex>
#include <thread>
#include <iostream>
#include <queue>
#include <chrono>

int main() {
  std::queue<int> produced_nums;
  std::mutex m;
  std::condition_variable cond_var;
  bool done = false;
  bool notified = false;
  std::thread producer([&]() {
    for (int i = 0; i < 5; ++i) {
      std::this_thread::sleep_for(std::chrono::seconds(1));
      std::unique_lock<std::mutex> lock(m);
      std::cout << "producing " << i << '\n';
      produced_nums.push(i);
      notified = true;
      cond_var.notify_one();
    }
```

Exemple 9 (STL) (suite)

```
    std::thread consumer([&]() {
        std::unique_lock<std::mutex> lock(m);
        while (!done) {
            while (!notified) {  // loop to avoid spurious wakeups
                cond_var.wait(lock);
            }
            while (!produced_nums.empty()) {
                std::cout << "consuming " << produced_nums.front() <<
                produced_nums.pop();
            }
            notified = false;
        }
    });

    producer.join();
    consumer.join();
}
```

OpenMP

- Bibliothèque de programmation parallèle
- Portable (fortran, C, C++)
- Utilise le modèle `Fork/Join`
- Simule la mémoire commune
- L'exécution commence avec le fil maître (1^{er} processus)
- Des régions parallèles démarrent de nouveaux fils
- Les fils se joignent à la fin de la région (seulement le fil maître poursuit son exécution)

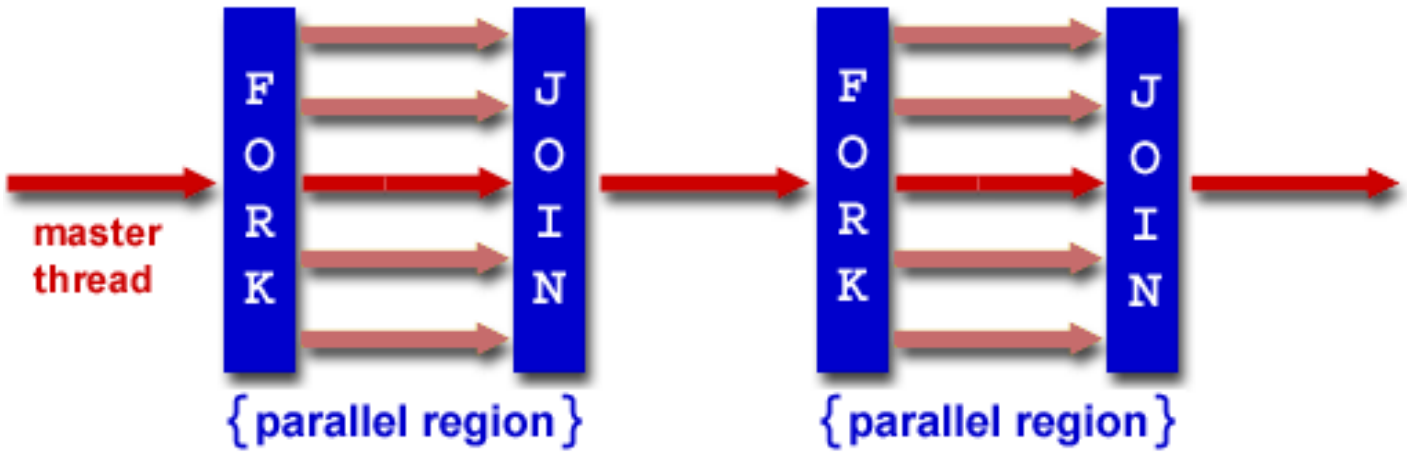
Composantes

- Trois composants primaires de l'API
  - Instructions au compilateur
  - Bibliothèque (environnement runtime)
  - Variables d'environnement

Historique

- 1997 - version 1.0 (fortran)
- 1998 - version 1.0 (C, C++)
- 2000 - version 2.0 (fortran)
- 2002 - version 2.0 (C, C++)
- 2008 - version 3.0
- 2011 - version 3.1

Modèle


master
thread
FORK
JOIN
{parallel region}
FORK
JOIN
{parallel region}

Modèle

- Plusieurs fils partagent de la mémoire
- Le parallélisme est explicite (fork/join)
- Le maître crée des équipes de fils parallèles
- les énoncés à exécuter sont englobés dans des régions

Syntaxe

`#pragma omp` *directive [clause ...] (newline)*

Syntaxe

```
#include <omp.h>
main ()  {

  int var1, var2, var3;
      ...
  execution sequentielle
      ...

  // Debut d'une region parallele (demarre un ensemble de fils)

  #pragma omp parallel private(var1, var2) shared(var3)
  {
    Section parallele executee par tous les fils
        ...
    Jonction de tous les fils, le maitre seul poursuit
  }
      ...
  poursuite de l'execution sequentielle
      ...
}
```

Syntaxe : directive "Parallel"

```
#include <omp.h>
main ()  {
   int nthreads, tid;

  /* demarrage de fils avec variables privees */
  #pragma omp parallel private(nthreads, tid)
  {
     /* on obtient le id du fil */
     tid = omp_get_thread_num();
     printf("Vous avez le bonjour du fil %d\n", tid);

     /* Seulement le fil maitre execute ce code */
     if (tid == 0)  {
        nthreads = omp_get_num_threads();
        printf("Number of threads = %d\n", nthreads);
     }
  }  /* jonction des fils */

}
```

Syntaxe : directive "Parallel"

- La maître a toujours le id 0
- Le nombre de fils dépend de la configuration (OMP_NUM_THREADS) (modifiable par la fonction `omp_set_num_threads`)

Syntaxe : directive de travail partagé

- Ne crée pas de nouveaux fils
- Il n'y a pas de barrière à l'entrée ni à la sortie
- 3 directives : for, section et single
- Elles doivent être associées à un énoncé `parallel`

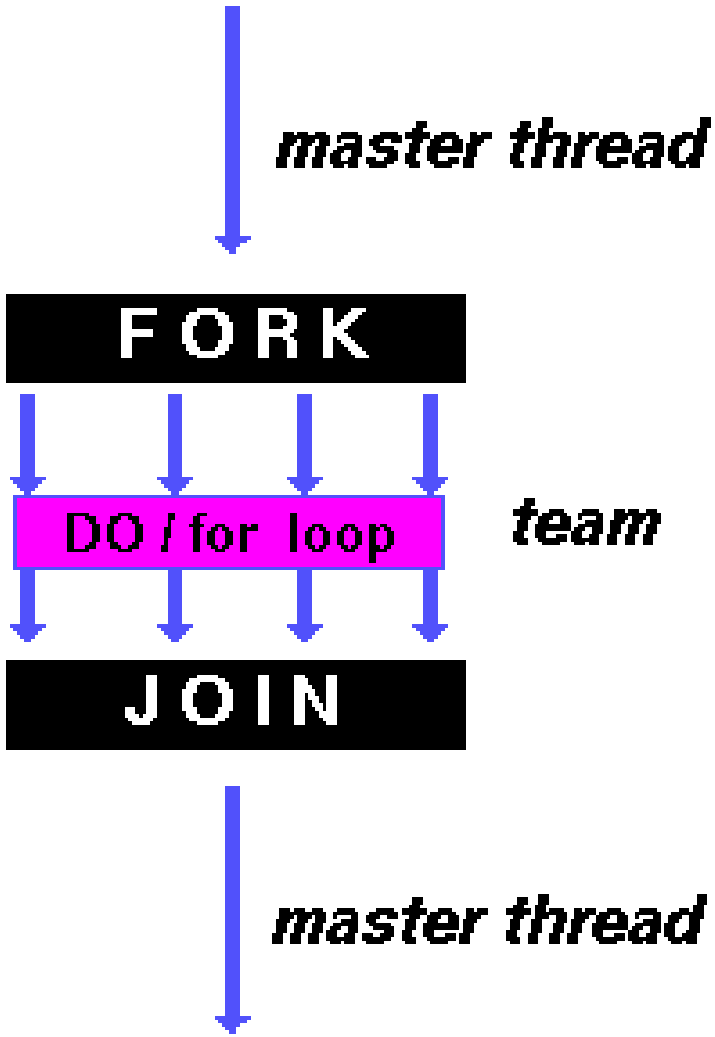
Syntaxe : directive "for"

```
#include <omp.h>
#define CHUNKSIZE 100
#define N     1000

main ()
{
  int i, chunk;
  float a[N], b[N], c[N];

  /* Some initializations */
  for (i=0; i < N; i++)   a[i] = b[i] = i * 1.0;
  chunk = CHUNKSIZE;

  #pragma omp parallel shared(a,b,c,chunk) private(i)
  {
     #pragma omp for schedule(dynamic,chunk) nowait
     for (i=0; i < N; i++)   c[i] = a[i] + b[i];
  }  /* end of parallel section */
}
```

Syntaxe : directive "for"

- `schedule` sert à indiquer le nombre d'éléments de la boucle associés à un même fil et le type d'association (dynamique ou statique)
- `nowait` indique aucune barrière à la fin de l'énoncé

Syntaxe : directive "for"



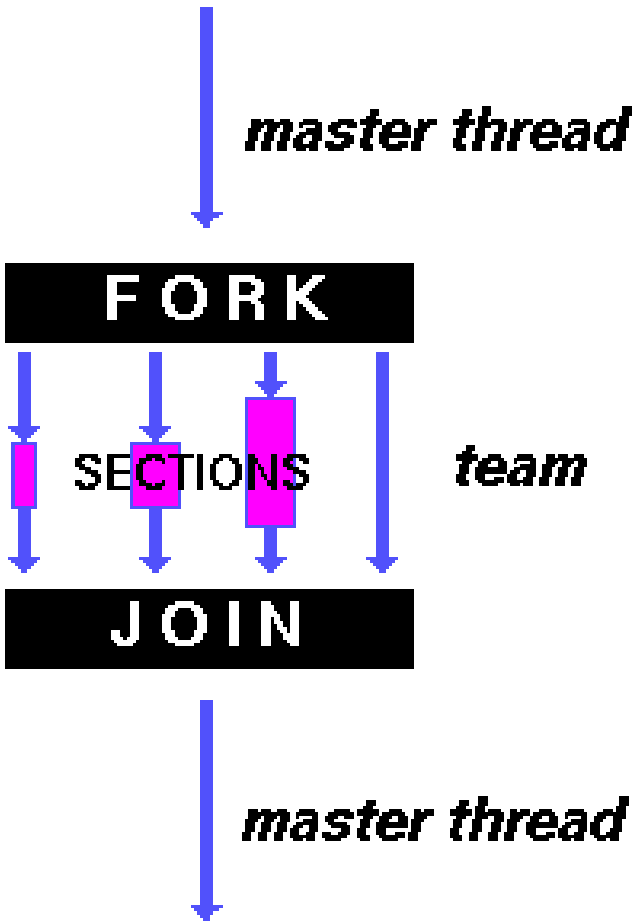
Syntaxe : directive "sections" et "section"

- L'énoncé `sections` est une construction non itérative composée d'énoncé `section`
- Les `sections` sont divisées entre les fils
- Des énoncés `section` peuvent sont imbriqués dans des énoncés `sections`
- Il y a une barrière à la fin d'un énoncé `sections` à moins d'indication contraire (nowait).

Syntaxe : directive "sections" et "section"

```
#include <omp.h>
#define N     1000
int main ()
{
   int i;
   float a[N], b[N], c[N];
   /* Some initializations */
   for (i=0; i < N; i++)  a[i] = b[i] = i * 1.0;

   #pragma omp parallel shared(a,b,c) private(i)
   {   #pragma omp sections nowait
       {
         #pragma omp section
         for (i=0; i < N/2; i++)  c[i] = a[i] + b[i];

         #pragma omp section
         for (i=N/2; i < N; i++)  c[i] = a[i] + b[i];

       }  /* end of sections */
   }  /* end of parallel section */
}
```

Syntaxe : directive "sections" et "section"



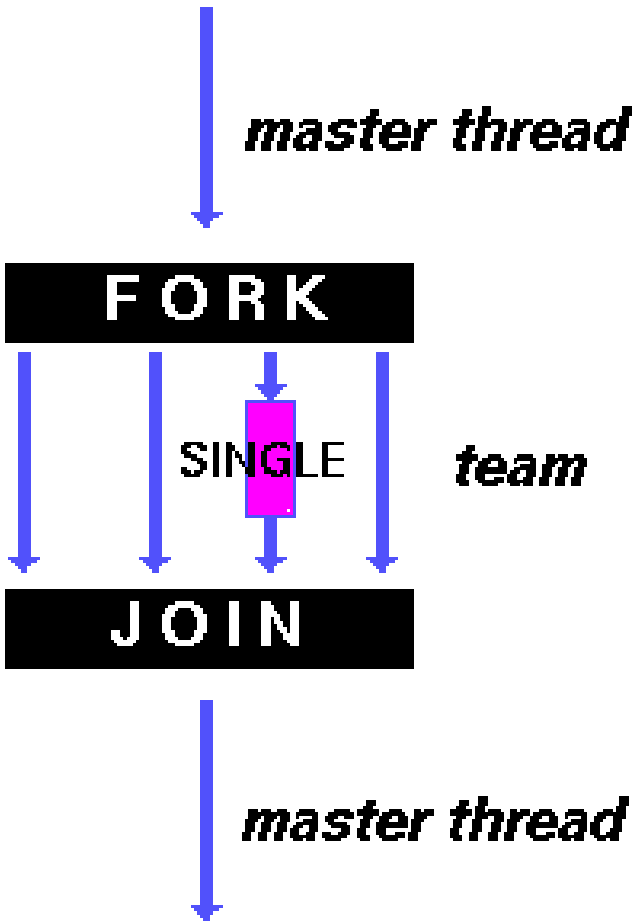
Syntaxe : directive "single"

- Le code visé est exécuté par un seul fil (exclusion mutuelle)

Syntaxe : directive "single"

```
#include <omp.h>
#define N     1000
int main ()
{
   #pragma omp parallel
   {
       fonction1();

       #pragma omp single
       {
          fonction2();
       }

       fonction3();

   }  /* end of parallel section */
}
```

Syntaxe : directive "single"



Syntaxe : directives combinées

- On peut combiner l'énoncé `parallel` et les directives de régions (`for`, `regions et single`).

Syntaxe : directives combinées

```
#include <omp.h>
#define N       1000
#define CHUNKSIZE   100

int main ()  {
   int i, chunk;
   float a[N], b[N], c[N];

   /* Some initializations */
   for (i=0; i < N; i++)  a[i] = b[i] = i * 1.0;
   chunk = CHUNKSIZE;

   #pragma omp parallel for shared(a,b,c,chunk) private(i) \
                            schedule(static,chunk)
     for (i=0; i < n; i++)
          c[i] = a[i] + b[i];
}
```

Syntaxe : Synchronisation "master"

- Spécifie une région qui ne sera exécutée que par le maître.
- Tous les autres fils sautent cette section de code.

```
#pragma omp master
{   section reservee au maitre }
```

Syntaxe : Synchronisation "critical"

- Assure l'exclusion mutuelle

```
#pragma omp critical [nom]
{   section critique }
```

Syntaxe : Synchronisation "critical"

- Assure l'exclusion mutuelle

```
#include <omp.h>

main()
{
   int x;
   x = 0;

   #pragma omp parallel shared(x)
   {
      #pragma omp critical
      x = x + 1;
   }  /* end of parallel section */
}
```

Syntaxe : Synchronisation "barrier"

- Synchronise tous les fils en un point

```
#pragma omp barrier
```

Syntaxe : Synchronisation "atomic"

- Spécifie qu'une zone en mémoire centrale doit être mise à jour de façon atomique

```
#pragma omp atomique          |  x++, ++x
{                             |  x--, --x
   operation                  |  x+= ..., x-= ...
}                             |  x*= ..., x\= ...
```

Syntaxe : Synchronisation "flush"

- Point de synchronisation ou la mémoire doit devenir cohérente
- Toutes les variables accessibles par un fil sont écrites en mémoire

Syntaxe : Synchronisation

- ordered – permet de sérialiser des itérations
- threadprivate – crée des variables privées à un fil.

Syntaxe : Portée des variables

- private (copyin)
- firstprivate, lastprivate
- shared
- default
- reduction (transforme private en shared dynamiquement)

Syntaxe : Directives

- OMP_Set_num_threads, OMP_Get_num_threads, OMP_Get_max_threads
- OMP_Get_thread_num
- OMP_in_parallel, OMP_set_dynamic, OMP_get_dynamic
- OMP_Set_nested, OMP_Get_nested
- OMP_init_lock, OMP_destroy_lock, OMP_set_lock, OMP_unset_lock, OMP_test_lock

Syntaxe :Variables systèmes

- OMP_SCHEDULE
- OMP_NUM_THREADS
- OMP_DYNAMIC
- OMP_NESTED

MPI - Message Passing Interface

- C'est une spécification
- Basée sur le passage de messages
- Le parallélisme est explicite
- Le nombre de tâches est statique (change dans MPI-2)

Structure d'un programme

```
#include <iostream>
#include <mpi.h>

using namespace std;

int main(int argc, char *argv[])
{
  int rank, size;

  MPI::Init(argc, argv);

  rank = MPI::COMM_WORLD.Get_rank();
  size = MPI::COMM_WORLD.Get_size();

  cout << "Je suis le " << rank << " de " << size << endl;

  MPI::Finalize();

  return 0;
}
```

Types de MPI

- MPI : :CHAR, MPI : :UNSIGNED_CHAR, MPI : :SIGNED_CHAR
- MPI : :SHORT, MPI : :INT, MPI : :LONG
- MPI : :UNSIGNED_SHORT, MPI : :UNSIGNED,
  MPI : :UNSIGNED_LONG
- MPI : :FLOAT, MPI : :DOUBLE, MPI : :LONG_DOUBLE
- MPI : :LONG_LONG, MPI : :UNSIGNED_LONG_LONG
- MPI : :WCHAR

Gestion de l'environnement

- MPI : :Init() — MPI_Init
- MPI : :Comm : :Get_size()— MPI_Comm_size
- MPI : :Comm : :Get_rank() — MPI_Comm_rank
- MPI : :Comm : :Abort() — MPI_Abort
- MPI : :Get_processor_name() — MPI_Get_processor_name
- MPI : :Finalize() — MPI_Finalize
- MPI : :Is_initialized() — MPI_Initialized

communication

- MPI fournit send/receive avec différents niveaux de synchronisation.
- Les messages peuvent être emmagasinés par le système ou l'application.
- Bloquant : retourne lorsqu'il est sécure de modifier le tampon contenant le message sans affecter récepteur
- Non-bloquant : retourne immédiatement (dangereux pour le tampon).
- Synchrone : Attend un ACK du récepteur.
- Asynchrone : retourne lorsque copié dans tampon système.
- Ordonnancement = FIFO
- Aucune équité n'est garantie.

communication

- Envoi bloquant
  C → MPI_Send(Tampon, taille, type, dest, tag, comm)
  C++ → MPI : :Comm : :Send(...)
- Envoi non-bloquant
  C → MPI_Isend(Tampon, taille, type, dest, tag, comm, etat)
  C++ → MPI : :Comm : :Isend(...)
- Réception bloquante
  C → MPI_Recv(Tampon, taille, type, dest, tag, comm, etat)
  C++ → MPI : :Comm : :Recv(...)
- Réception non-bloquante
  C → MPI_Irecv(Tampon, taille, type, dest, tag, comm, request)
  C++ → MPI : :Comm : :Irecv() const

communication

MPI : :Comm : :Bsend()
MPI : :Comm : :Ssend()
MPI : :Comm : :Rsend()
MPI : :Attach_buffer()
MPI : :Detach_buffer()
MPI : :Comm : :Ibsend()
MPI : :Comm : :Issend()
MPI : :Comm : :Irsend()
MPI : :Comm : :Sendrecv()

communication

MPI : :Request : :Wait()
MPI : :Request : :Test()
MPI : :Request : :Waitany()
MPI : :Request : :Testany()
MPI : :Request : :Waitall()
MPI : :Request : :Testall()
MPI : :Request : :Waitsome()
MPI : :Request : :Testsome()
MPI : :Comm : :Iprobe()
MPI : :Comm : :Probe()
MPI : :Request : :Cancel()
MPI : :Status : :Is_cancelled()

communication

MPI : :Intracomm : :Barrier()
MPI : :Intracomm : :Bcast()
MPI : :Intracomm : :Gather()
MPI : :Intracomm : :Scatter()
MPI : :Intracomm : :Allgather()
MPI : :Intracomm : :Alltoall()
MPI : :Intracomm : :Reduce()
MPI : :Op : :Init()
MPI : :Op : :Free()
MPI : :Intracomm : :Allreduce()
MPI : :Intracomm : :Reduce_scatter()
MPI : :Intracomm : :Scan()

Opérations sur les groupes

MPI : :Comm : :Get_group()
MPI : :Group : :Union()
MPI : :Group : :Intersect()
MPI : :Group : :Difference()
MPI : :Group : :Incl()
MPI : :Group : :Excl()
MPI : :Group : :Range_incl()
MPI : :Group : :Range_excl()
MPI : :Comm : :Get_size()
MPI : :Comm : :Get_rank()

Opérations sur les groupes

MPI : :Comm : :Compare()
MPI : :Intracomm : :Dup()
MPI : :Comm : :Clone()
MPI : :Intracomm : :Create()
MPI : :Intracomm : :Split()
MPI : :Comm : :Is_inter()
MPI : :Intracomm : :Create_intercomm()
MPI : :Intercomm : :Merge()

Topologie virtuelle

MPI : :Intracomm : :Create_cart()
MPI : :Intracomm : :Create_graph()
MPI : :Comm : :Get_topology()
MPI : :Graphcomm : :Get_dims()
MPI : :Graphcomm : :Get_topo()
MPI : :Cartcomm : :Get_dim()
MPI : :Cartcomm : :Get_topo()
MPI : :Cartcomm : :Get_cart_rank()
MPI : :Cartcomm : :Get_coords()
MPI : :Graphcomm : :Get_neighbors_count()
...

Exemple

```
#include <math.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  int n, rank, size, i;
  double PI25DT = 3.14159265358979323846264643;
  double mypi, pi, h, sum, x;

  MPI::Init(argc, argv);
  size = MPI::COMM_WORLD.Get_size();
  rank = MPI::COMM_WORLD.Get_rank();
}
```

Exemple

```
  while (1) {
     if (rank == 0) {
        cout << "Enter the number of intervals:(0 quits)"<<endl;
        cin >> n;
     }
     MPI::COMM_WORLD.Bcast(&n, 1, MPI::INT, 0);
     if (n==0)   break;
     else {
       h = 1.0 / (double) n;
       sum = 0.0;
       for (i = rank + 1; i <= n; i += size) {
           x = h * ((double)i - 0.5);
           sum += (4.0 / (1.0 + x*x));
       }
       mypi = h * sum;
       MPI::COMM_WORLD.Reduce(&mypi, &pi, 1, MPI::DOUBLE,
                              MPI::SUM, 0);
       if (rank == 0)
           cout << "pi is approximately " << pi
                << ", Error is " << fabs(pi - PI25DT) << endl;
     }
  }
  MPI::Finalize();
  return 0;
}
```

Exemple

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
int main( int argc, char *argv[] )
{
    int n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
```

Exemple

```
while (1) {
   if (myid == 0) {
       printf("Enter the number of intervals: (0 quits) ");
       scanf("%d",&n);
   }
   MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
   if (n == 0)  break;
   else {
      h   = 1.0 / (double) n;
      sum = 0.0;
      for (i = myid + 1; i <= n; i += numprocs) {
          x = h * ((double)i - 0.5);
          sum += (4.0 / (1.0 + x*x));
      }
      mypi = h * sum;
      MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
                 MPI_COMM_WORLD);
      if (myid == 0)
          printf("pi is approximately %.16f, Error is %.16f\n",
                 pi, fabs(pi - PI25DT));
   }
}
MPI_Finalize();
return 0;
}
```

Exemple

```
#include <stdio.h>
#include "mpi.h"

int main(int  argc, char **argv )
{
    int rank, size;
    MPI_Comm new_comm;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_split( MPI_COMM_WORLD, rank == 0, 0, &new_comm );
    if (rank == 0)
        master_io( MPI_COMM_WORLD, new_comm );
    else
        slave_io( MPI_COMM_WORLD, new_comm );

    MPI_Finalize( );
    return 0;
}
```

Exemple

```
/* This is the master */
int master_io( master_comm, comm )
MPI_Comm comm;
{
    int         i,j, size;
    char        buf[256];
    MPI_Status status;

    MPI_Comm_size( master_comm, &size );
    for (j=1; j<=2; j++) {
        for (i=1; i<size; i++) {
            MPI_Recv( buf, 256, MPI_CHAR, i, 0,
                                master_comm, &status );
            fputs( buf, stdout );
        }
    }
}
```

Exemple

```
/* This is the slave */

int slave_io( master_comm, comm )
MPI_Comm comm;
{
   char buf[256];
   int  rank;

   MPI_Comm_rank( comm, &rank );
   sprintf( buf, "Hello from slave %d\n", rank );
   MPI_Send(buf, strlen(buf)+1, MPI_CHAR, 0, 0, master_comm);

   sprintf( buf, "Goodbye from slave %d\n", rank );
   MPI_Send(buf, strlen(buf)+1, MPI_CHAR, 0, 0, master_comm);

   return 0;
}
```

Exemple

In this example, you will put together some of the previous examples to implement a simple Jacobi iteration for approximating the solution to a linear system of equations.
In this example, we solve the Laplace equation in two dimensions with finite differences. This may sound involved, but really amount only to a simple computation, combined with the previous example of a parallel mesh data structure.

Exemple

```
#include <stdio.h>
#include <math.h>
#include "mpi.h"
/*This example handles a 12 x 12 mesh,on 4 processors only.*/
#define maxn 12
int main( argc, argv )
int argc;
char **argv;
{
   int         rank, value, size, errcnt, toterr, i, j, itcnt;
   int         i_first, i_last;
   MPI_Status status;
   double      diffnorm, gdiffnorm;
   double      xlocal[(12/4)+2][12];
   double      xnew[(12/3)+2][12];

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size( MPI_COMM_WORLD, &size );
```

Exemple

```
if (size != 4) MPI_Abort( MPI_COMM_WORLD, 1 );

/*xlocal[][0] is lower ghostpoints, xlocal[][maxn+2] is
  upper */
/*Note that top and bottom processes have one less row of
  interior points */
i_first = 1;
i_last  = maxn/size;
if (rank == 0)          i_first++;
if (rank == size - 1) i_last--;

/* Fill the data as specified */
for (i=1; i<=maxn/size; i++)
    for (j=0; j<maxn; j++)
        xlocal[i][j] = rank;
for (j=0; j<maxn; j++) {
    xlocal[i_first-1][j] = -1;
    xlocal[i_last+1][j] = -1;
}
```

Exemple

```
itcnt = 0;
do {
  /*Send up unless I'm at the top, then receive from below*/
  /*Note the use of xlocal[i] for &xlocal[i][0] */
  if (rank < size - 1)
     MPI_Send(xlocal[maxn/size], maxn, MPI_DOUBLE, rank+1,
                    0, MPI_COMM_WORLD );
  if (rank > 0)
     MPI_Recv( xlocal[0], maxn, MPI_DOUBLE, rank - 1, 0,
                   MPI_COMM_WORLD, &status );
  /* Send down unless I'm at the bottom */
  if (rank > 0)
     MPI_Send( xlocal[1], maxn, MPI_DOUBLE, rank - 1, 1,
                   MPI_COMM_WORLD );
  if (rank < size - 1)
     MPI_Recv(xlocal[maxn/size+1], maxn, MPI_DOUBLE, rank+1,
                    1, MPI_COMM_WORLD, &status );
```

Exemple

```
        /* Compute new values (but not on boundary) */
        itcnt ++;
        diffnorm = 0.0;
        for (i=i_first; i<=i_last; i++)
           for (j=1; j<maxn-1; j++) {
               xnew[i][j] = (xlocal[i][j+1] + xlocal[i][j-1] +
                             xlocal[i+1][j] + xlocal[i-1][j])/4.0;
               diffnorm += (xnew[i][j] - xlocal[i][j]) *
                           (xnew[i][j] - xlocal[i][j]);
           }
        /* Only transfer the interior points */
        for (i=i_first; i<=i_last; i++)
            for (j=1; j<maxn-1; j++)
                xlocal[i][j] = xnew[i][j];
        MPI_Allreduce( &diffnorm, &gdiffnorm, 1, MPI_DOUBLE,
                       MPI_SUM, MPI_COMM_WORLD );
        gdiffnorm = sqrt( gdiffnorm );
        if (rank == 0) printf( "At iteration %d, diff is %e\n",
                               itcnt, gdiffnorm );
    } while (gdiffnorm > 1.0e-2 && itcnt < 100);
    MPI_Finalize( );
    return 0;
}
```

Message Oriented Middelware

All good DAD need a good MOM...

(Distributed Application Deployments)

Message Oriented Middelware

Un MOM est un logiciel de type serveur dont le rôle est de gérer les échanges de messages entre différents types d'applications.

MOM

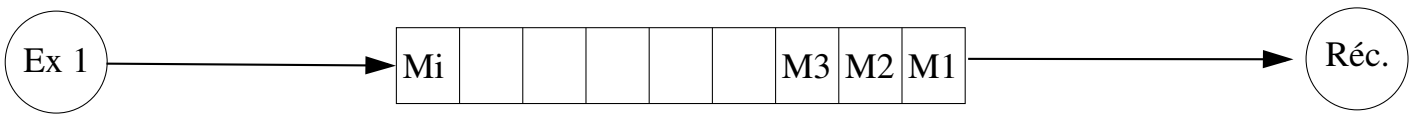
Un MOM utilise :

- Un courtier de message (message Broker)
- Des files de messages
- Différents modes de communication
- Différents modes de gestion

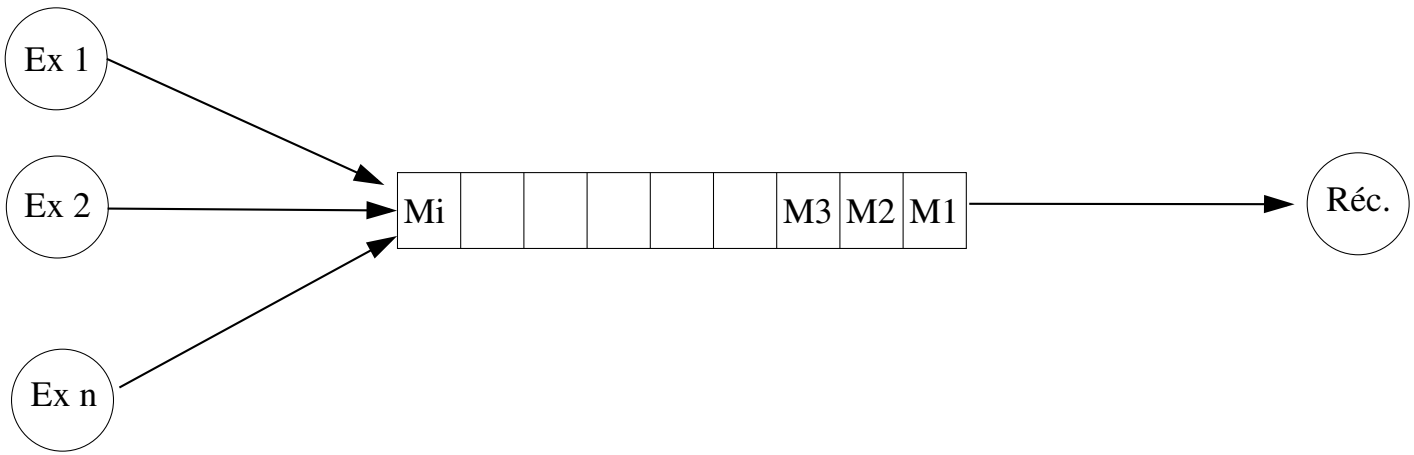
MOM - Modes de communication et services

- Asynchrone/synchrone
- Réception bloquante/destructrice ou non
- Point à point ou publication/souscription
- Fifo/priorité
- Notification
- Exactly once/at most once/at least once
- Sécurité et filtrage
- Différents formats (SOAP, XML, texte)
- Transaction

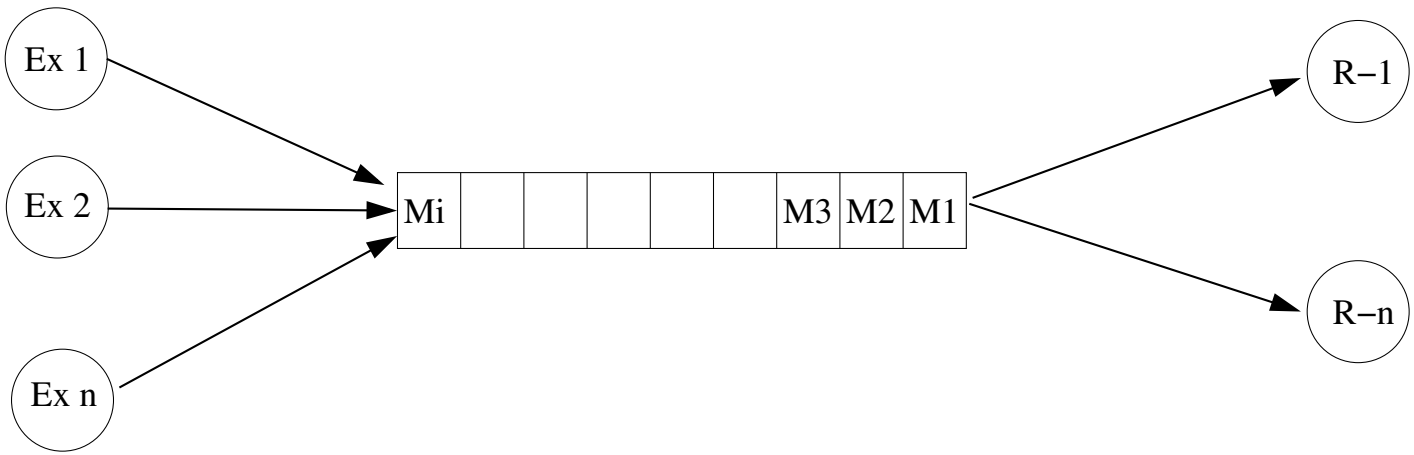
MOM - un à un



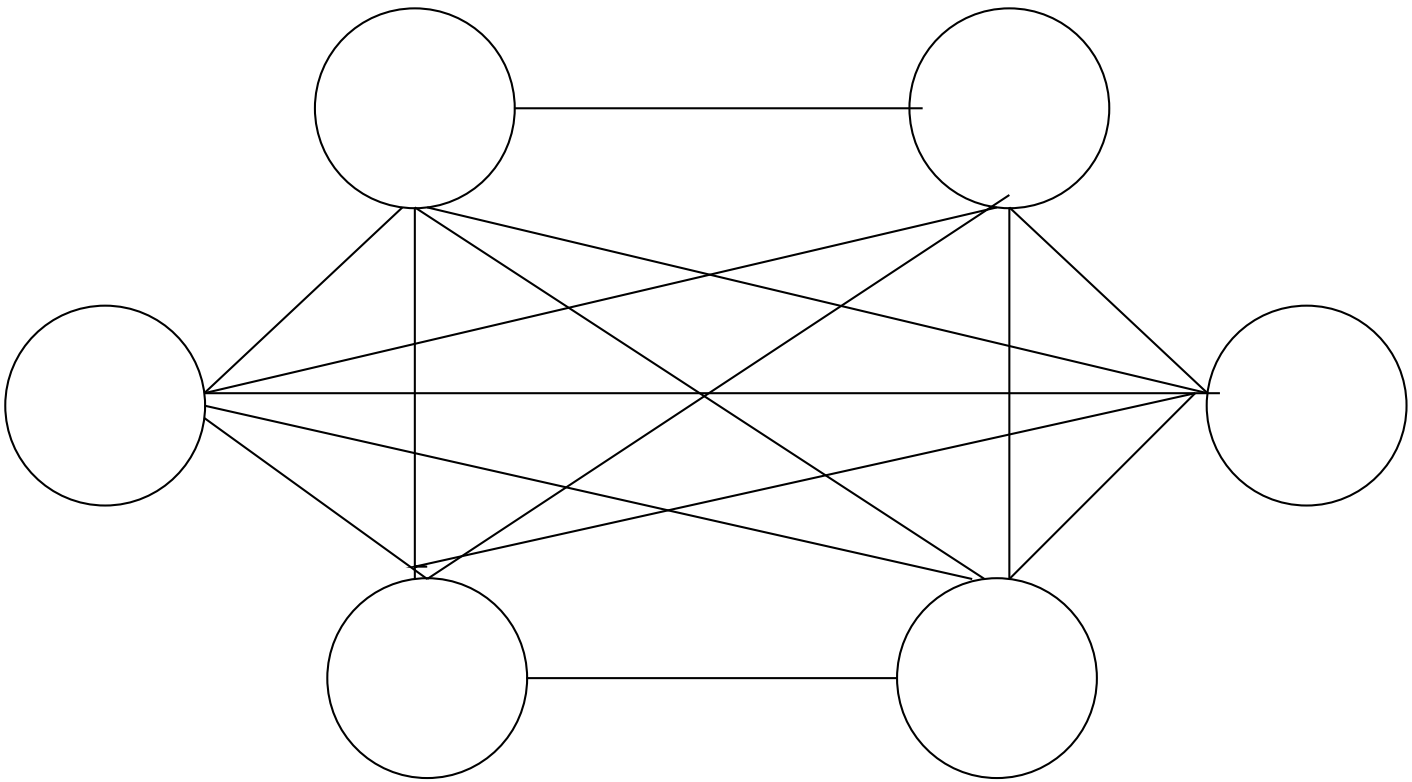
MOM - plusieurs à un



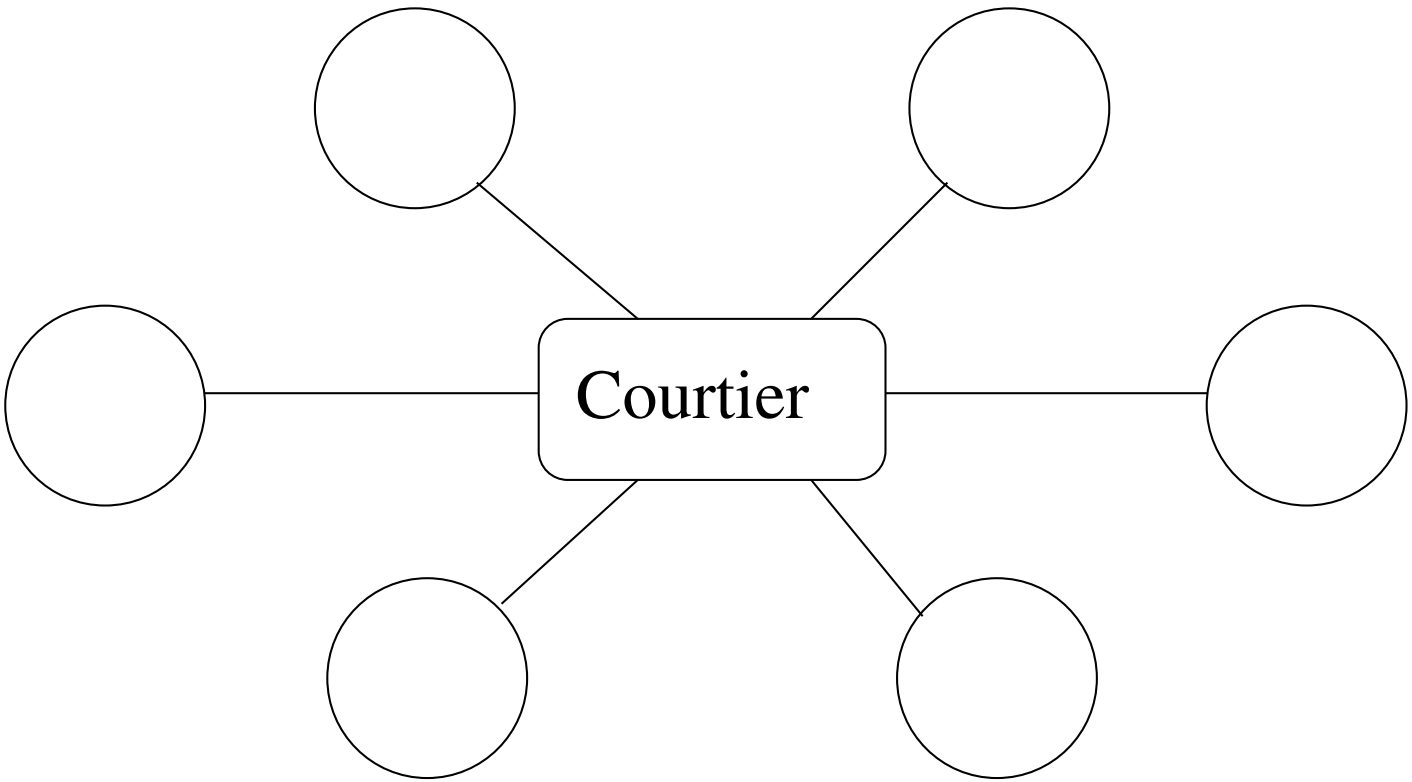
MOM - plusieurs à plusieurs



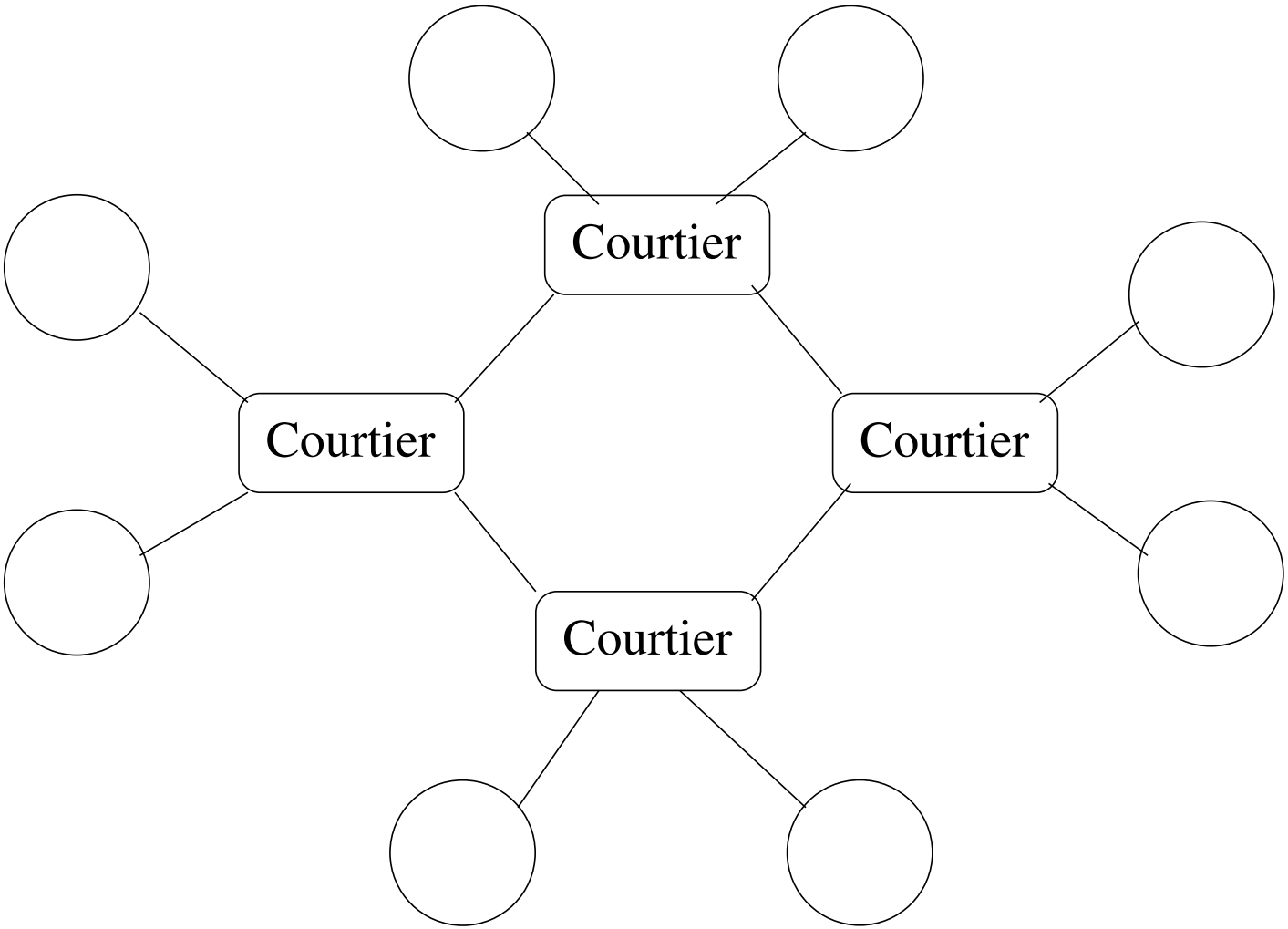
MOM - connexions



MOM - connexions



MOM - connexions



MOM - Comparaison

- MOM vs RPC
- MOM vs Courriel

MOM - Exemples de systèmes et normes

- Open JMS (Sun-Oracle : plus une spécification pour Java)
- AMQP (spécification)
- MQTT (spécification gérée par OASIS)
- IBM Websphere MQ (Anciennement MQ series)
- Sonic MQ (Sonic Software)
- MSMQ (Microsoft)
- Apache Active MQ ((Apache project Geronimo)
- Queue Manager Joram
- MQTT

MOM - Domaine d'utilisation

Les MOM sont sutout utilisés pour :

- EAI : Enterprise Application Integration
- ESB : Enterprise Service Bus
- Data warehouse
- Messagerie inter-bancaire
- Diffusion d'information

MOM - Exemple : JMS

La spécification définie :

- Une interface pour un fournisseur de service
- un API

MOM - Exemple : JMS

API :

- createQueue/createSender/createReceiver
- createQueueConnection
- createTextMessage/setText/Send
- Receive (bloquant et non bloquant)
- msgListener (notify)

Chapitre 3/4/5/6 - Parallélisme/Étude de cas

1. Langages
   - CSP
   - Ada
   - Java
   - Python
   - C#
   - C++
   - C++
   - OpenMP
   - MPI
   - MOM
2. Systèmes d'exploitation
   - Unix
   - Solaris
   - Windows
   - Posix

Processus

- Fork - relation parent (pid)/enfant (0)
- Wait (signal) : attend la fin de l'exécution et les résultats
- Synchronisation : sémaphores
- communication : message, pipe, fichiers

Processus

```
#include <stdio.h>
#include <sys/wait.h>
#include <sys/signal.h>
#include <errno.h>
extern "C" {
   int fork();
   int execl(char *, int);
   int kill(int , int);
   int wait(void *);
}
class Pcs {
   private:
      int pcsid;
      int status;
   public:
      Pcs();
      Pcs(char *fichier);
      Fork(char *fichier);
      Join();
      Kill();                    };
```

Exemple

```
Pcs::Pcs() {}

Pcs::Pcs(char *fichier)
{
   if ((pcsid = fork()) == -1)
      perror("Classe Pcs Function Fork (Fork)");
   else
     if (pcsid == 0)
       if (execl(fichier,0)== -1)
          perror("Classe Pcs Function Fork (exec)");
}
Pcs::Fork(char *fichier)
{
   if ((pcsid = fork()) == -1)
      perror("Classe Pcs Function Fork (Fork)");
   else
     if (pcsid == 0)
       if (execl(fichier,0)== -1)
          perror("Classe Pcs Function Fork (exec)");
}
```

Exemple

```
Pcs::Kill()
{
  if (kill(pcsid, SIGKILL) == -1)
     perror("Classe Pcs Function Kill");
}

Pcs::Join()
{
   if ((pcsid = wait(&status)) == -1)
      perror("Classe Pcs Function Join");
}
```

Mémoire

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <errno.h>
extern "C" {
   int shmget(key_t, int, int);
   char * shmat(int, char *, int);
   int shmdt(char *);
   int shmctl(int, int);
}
class Mem {
   private:
        int memid;
   public:
         Mem(key_t key, int dimension);
         void Attache(char **adresse);
         void Detache(char *adresse);
         Detruit();
};
```

Exemple

```
Mem::Mem(key_t key, int dimension)
{
   if ((memid = shmget(key, dimension, IPC_CREAT|0666)) == -1)
      perror("Classe Mem constructeur");
}
void Mem::Detache(char *adresse)
{
   if (shmdt(adresse) == -1)
      perror("Classe Mem function Detache");
}
void Mem::Attache(char **adresse)
{
   *adresse = NULL;
   if ((*adresse = shmat(memid, *adresse, 0)) == (char *) -1)
      perror("Classe Mem function Attache");
}
void Mem::Detruit()
{
   if (shmctl(memid, IPC_RMID) == -1)
      perror("Classe Mem Destructeur");
}
```

Communication pas messages

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>
extern "C" {
   int msgget(key_t,int);
   int msgctl(int, int, struct msqid_ds *);
   int msgsnd(int, struct msgbuf *, int, int);
   int msgrcv(int, struct msgbuf *, int, long, int);
}
class Msg {
   private:
      int msgqid;
   public:
      Msg(key_t key);
      Envoie(void *message, int dim);
      Recoit(void *message, int dim);
      Detruit();
};
```

Exemple

```
Msg::Msg(key_t key)
{
   if ((msgqid = msgget(key, IPC_CREAT|IPC_EXCL|0666)) == -1)
      if ((msgqid = msgget(key, IPC_CREAT|0666)) == -1)
         perror("Classe Msg Construteur");
}
Msg::Detruit()
{
   if (msgctl(msgqid, 0, IPC_RMID) == -1)
      perror("Classe Msg Destructeur");
}
Msg::Envoie(void *message, int dim)
{
   if (msgsnd(msgqid, message, dim, 0) == -1)
      perror("Classe Msg Function Envoie");
}
Msg::Recoit(void *message, int dim)
{
   if (msgrcv(msgqid, message, dim, 0, 0) == -1)
      perror("Classe Msg Function Recoit");
}
```

Sémaphore

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>

#define P_OP -1
#define V_OP 1

extern "C" {
    int semget(key_t, int, int);
    int semctl(int, int, int, semun );
    int semop(int, struct sembuf *, int);
}
```

Sémaphore

```
class Sem {
   private:
      int semid;
   public:
      Sem();
      Sem(key_t key,int valeur);
      void Init(key_t key,int valeur);
      void P();
      void V();
      Detruit();
};
```

Exemple

```
Sem::Sem(key_t key,int valeur)
{
  union semun arg;
  arg.val = valeur;

  if ((semid=semget(key, 1, IPC_CREAT|IPC_EXCL|0666)) == -1)
     if ((semid = semget(key, 1, IPC_CREAT|0666)) == -1)
       perror("Classe Sem Constructeur erreur a la creation");
  else
     if (semctl(semid, 0, SETVAL, arg) == -1)
        perror("Classe Sem Construct. erreur initialisation");
}

Sem::Sem()
{}
```

Exemple

```
void Sem::Init(key_t key,int valeur)
{
  union semun arg;
  arg.val = valeur;

  if ((semid=semget(key, 1, IPC_CREAT|IPC_EXCL|0666)) == -1)
     if ((semid = semget(key, 1, IPC_CREAT|0666)) == -1)
       perror("Classe Fct Init erreur a la creation");
  else
     if (semctl(semid, 0, SETVAL, arg) == -1)
        perror("Classe Fct Init erreur initialisation");
}
```

Exemple

```
Sem::Detruit()
{
   union semun arg;
      if (semctl(semid, 0, IPC_RMID, arg) == -1)
         perror("Classe Sem Destructeur");
}
```

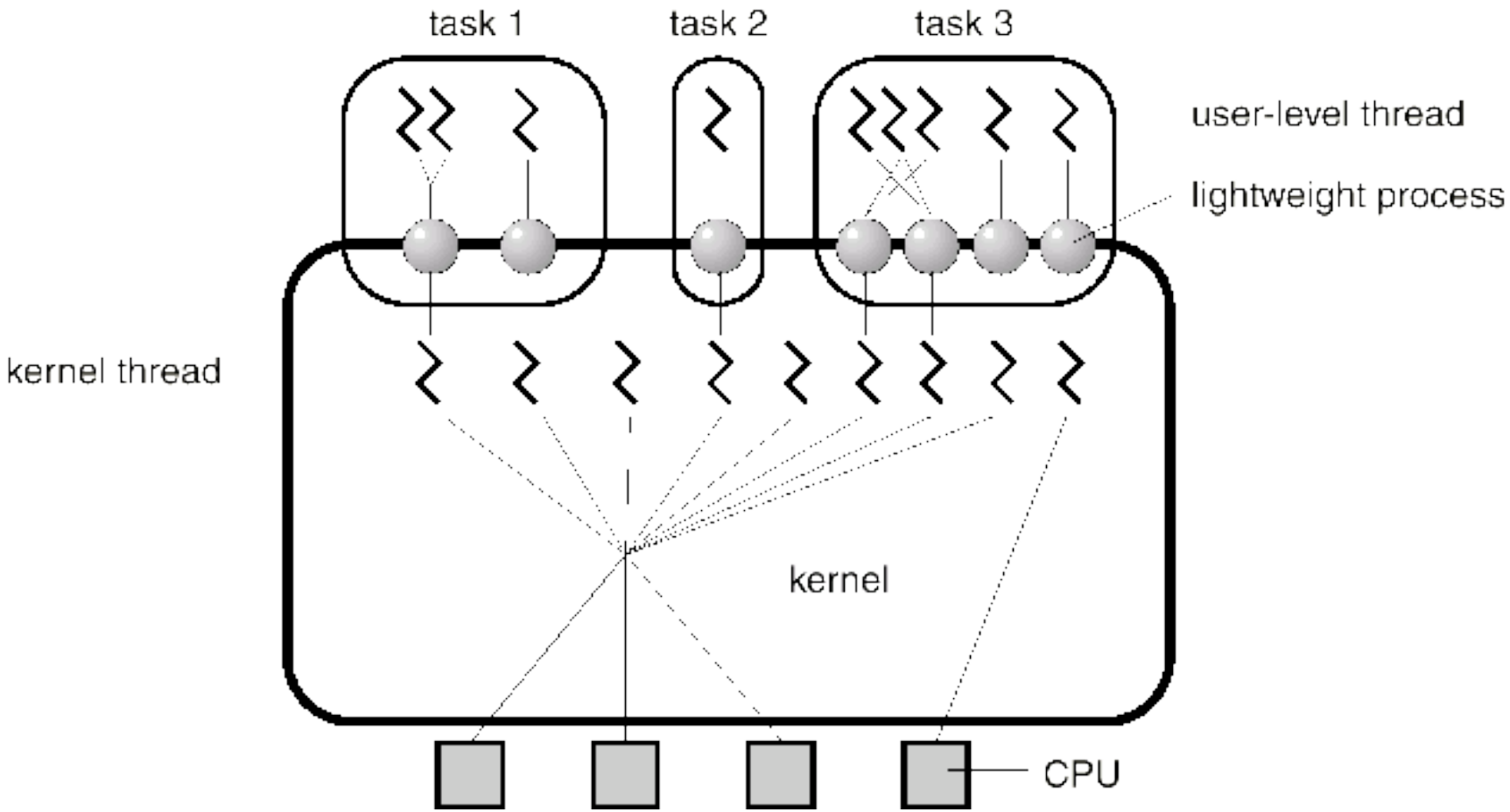
Exemple

```
void Sem::P()
{
   struct sembuf cmd;
   cmd.sem_num = 0;
   cmd.sem_op  = P_OP;
   cmd.sem_flg = 0;

   if (semop(semid, &cmd, 1) == -1)
      perror("Classe Sem function P");
}
void Sem::V()
{
   struct sembuf cmd;
   cmd.sem_num = 0;
   cmd.sem_op  = V_OP;
   cmd.sem_flg = 0;

   if (semop(semid, &cmd, 1) == -1)
      perror("Classe Sem function V");
}
```

Solaris – fil (thread)

- Il y a les fil usager et noyau
- Il y a les ressources d'exécution (lightweight process)
- Les fils peuvent se partager un LWP ou être attaché à un LWP

Solaris – fil (thread)



Solaris – fil (thread)

- thr_create, thr_exit, thr_self, thr_join
- exit
- thr_setconcurrency, thr_set_prio, thr_getprio
- thr_yield
- thr_sigsetmask, thr_kill
- thr_keycreate, thr_setspecific, thr_getspecific

Solaris – fil (Synchronisation)

- mutex_(init/destroy/lock/trylock/unlock)
- cond_(init/destroy/wait/signal/broadcast/timedwait)
- sema_(init/destroy/wait/trywait/post)
- rwlock_(init/destroy/rdlock/tryrdlock)
- rwlock_(wrlock/trywrlock/unlock)

Solaris – processus vs thread

- fork : copie tout le pcs (fils et lwp)
- fork1 : copie pcs avec un seul fil et LWP
- vfork copie un seul fil (pas l'espace d'adresses)
- exec : comme fork1

Solaris – Temps de création

- fork : 1700 micro-secondes sur un SS2
- fil non attaché (unbound) : 52 micro-secondes sur un SS2
- fil attaché (bound) : 350 micro-secondes sur un SS2

Solaris – multi-processur

swap et attente active

Windows – processus et fil

- CreateProcess, (NTCreateProcess + NTCreateThread)
- CreateThread, _beginThread, _beginThreadex
- CreateFiber
- ExitProcess (terminateProcess)
- ExitThread, ExitFiber
- SetPriorityClass, SetThreadPriority, GetCurrentThreadId

Windows – Synchronisation et communication

- fichiers, mémoire partagée, pipe, socket
- mutex, semaphore, event, critical section

Windows – Synchronisation et communication

- CreateMutex, ReleaseMutex, OpenMutex
- CreateSemaphore, ReleaseSemaphore, OpenSemaphore
- InitializeCriticalSection, EnterCriticalSection, leaveCriticalSection, DeleteCriticalSection
- PulseEvent (SetEvent, ResetEvent)
- WaitForMultipleObjects, WaitForSingleObjects, CloseHandle

Windows – multi-processeurs

attente active

Posix – fil

- pthread_create, pthread_exit, pthread_self, pthread_equal, pthread_join
- pthread_attr_init, pthread_attr_setdetachstate, pthread_attr_getdetachstate, pthread_attr_destroy
- pthread_detach

Posix – synchronisation

- pthread_mutex_(init/destroy/lock/unlock/trylock), pthread_mutex_attr_(init/destroy)
- pthread_cond_(init/destroy/wait/signal/broadcast), pthread_cond_attr_(init/destroy)