Introduction à la programmation concurrente

Introduction

Yann Thoma, Jonas Chapuis

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Février 2018

Programmation standard

Définition

- Un processus est une entité active et exécutable
  ou
- Un processus est un programme en cours d'exécution

- Un programme exécute des instructions séquentiellement
- Le développeur maîtrise la suite des opérations
- L'exécution est prévisible
- L'exécution est reproductible

Qu'est-ce que la programmation concurrente?

- La programmation concurrente est un paradigme de programmation tenant compte, dans un programme, de plusieurs contextes d'exécution (threads, processus, tâches) matérialisés par une pile d'exécution (stack) et des données privées
- La concurrence est indispensable lorsque l'on souhaite écrire des programmes interagissant avec le monde réel ou tirant parti de multiples processeurs (multi-coeurs, clusters, cloud, ...)
- Un processus est décomposé en *threads*
- Un thread est une sorte de *processus léger*
- Un thread correspond à une tâche qui s'exécute
  - Plus ou moins indépendamment des autres

Pourquoi la programmation concurrente?

- Optimiser l'utilisation du/des processeurs
- Eviter de bloquer sur des entrées/sorties (IO)
- Tirer avantage des architectures multi-cores :
  - Aujourd'hui n'importe quel PC possède au moins 2 coeurs
- Augmenter le parallélisme :
  - Tout programme faisant du calcul devrait être développé de manière concurrente
- Attendre des événements de plusieurs entrées
- Simplifier la structure d'un programme
- Satisfaire des contraintes temporelles :
  - Programmation temps réel

Exemple d'applications multi-thread

- Serveur web ou ftp : chaque client est géré par un thread
- Browser : un thread gère une connexion, un thread fait le rendu de la page, un thread décode une vidéo, un thread gère l'interaction avec l'interface (tout ceci pour un seul onglet)
- Jeu vidéo : un thread s'occupe du rendu graphique, un autre de l'IA, un autre du son, n threads gèrent n joypads, etc.
- *Viewer* d'images : un thread affiche l'image courante alors que n autres threads chargent les n images suivantes
- Programme de retouche d'image : image divisée en n blocs, n threads faisant le rendu de 1 bloc

Problèmes

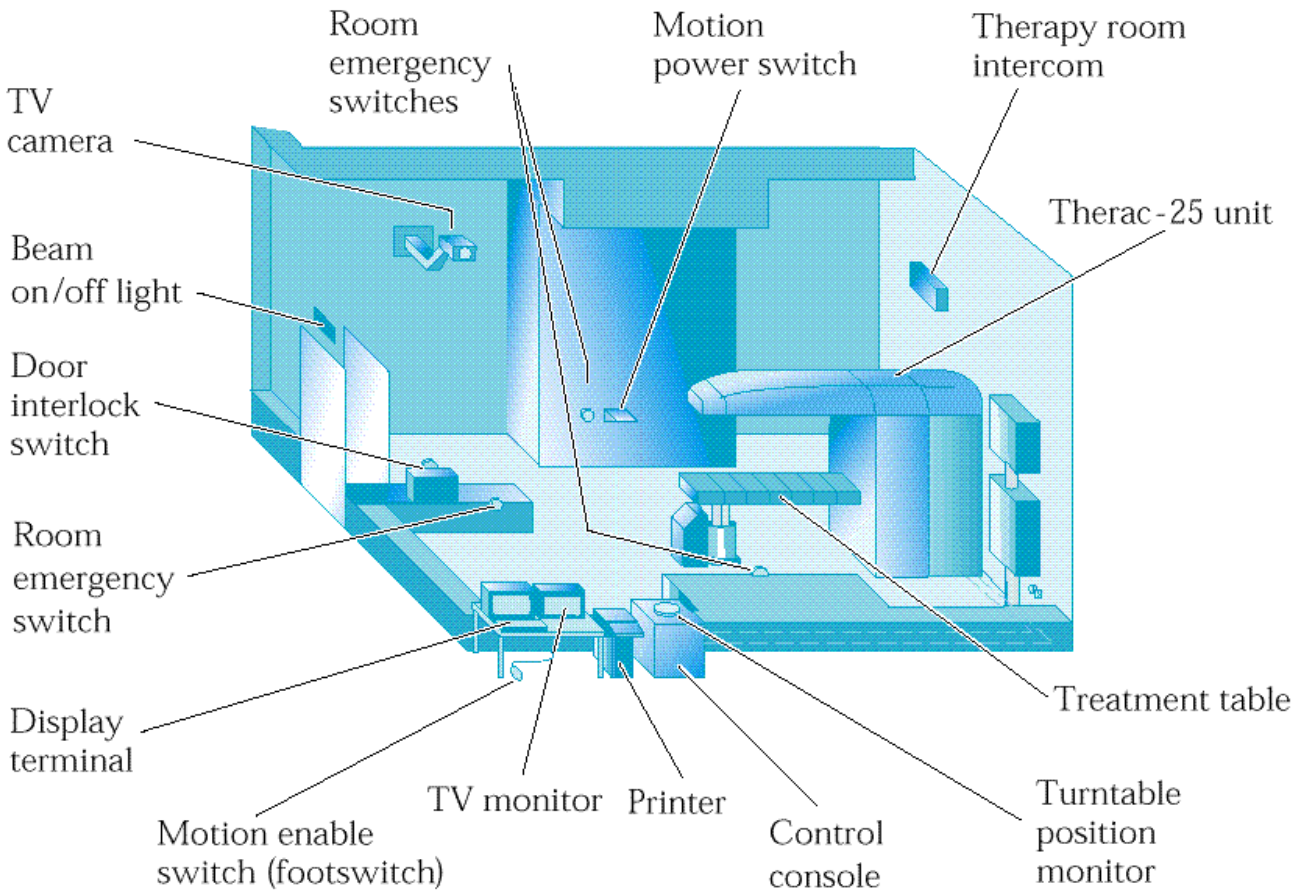
- Difficulté à synchroniser des tâches
- Gestion des ressources partagées
- Problème de predictibilité
- Problème de reproductibilité
- Concrètement:
  - Thérapie par radiation: Therac-25, entre 85 et 87
  - Blackout au Nord-Est des US en 2003

Problèmes - Blackout

- Blackout au Nord-Est des US en 2003 (55 millions de personnes touchées)

Problèmes - Radiations

- Thérapie par radiation: Therac-25, 6 accidents entre 85 et 87



Programmation parallèle vs. concurrente

- *Programmation parallèle (ou répartie)*
  - Des processus s'exécutent sur plusieurs processeurs
- *Programmation concurrente*
  - Les tâches sont gérées par un même processeur
- Les mécanismes de synchronisation sont différents

Question

- Soient les deux tâches suivantes (s'exécutant en pseudo-parallèle):

Tâche A

```
x = 3;

printf("%d",x);
```

Tâche B

```
x = 5;
```

- Quelle est la sortie du programme dans le terminal ?
- Que vaut `x` ?

Thread

- Un thread est un fil d'exécution
- Un processus est composé de plusieurs threads
- Les threads s'exécutent en "*parallèle*"
  - Sur un mono-processeur: chacun son tour
    - Par entrelacement
  - Sur un multi-processeur: peut être réellement parallèle
- Dans tous les cas, l'ordonnanceur est responsable de l'ordre d'exécution
- Problèmes:
  - Accès à des ressources partagées
  - Echange de données
  - Séquentialité de l'exécution

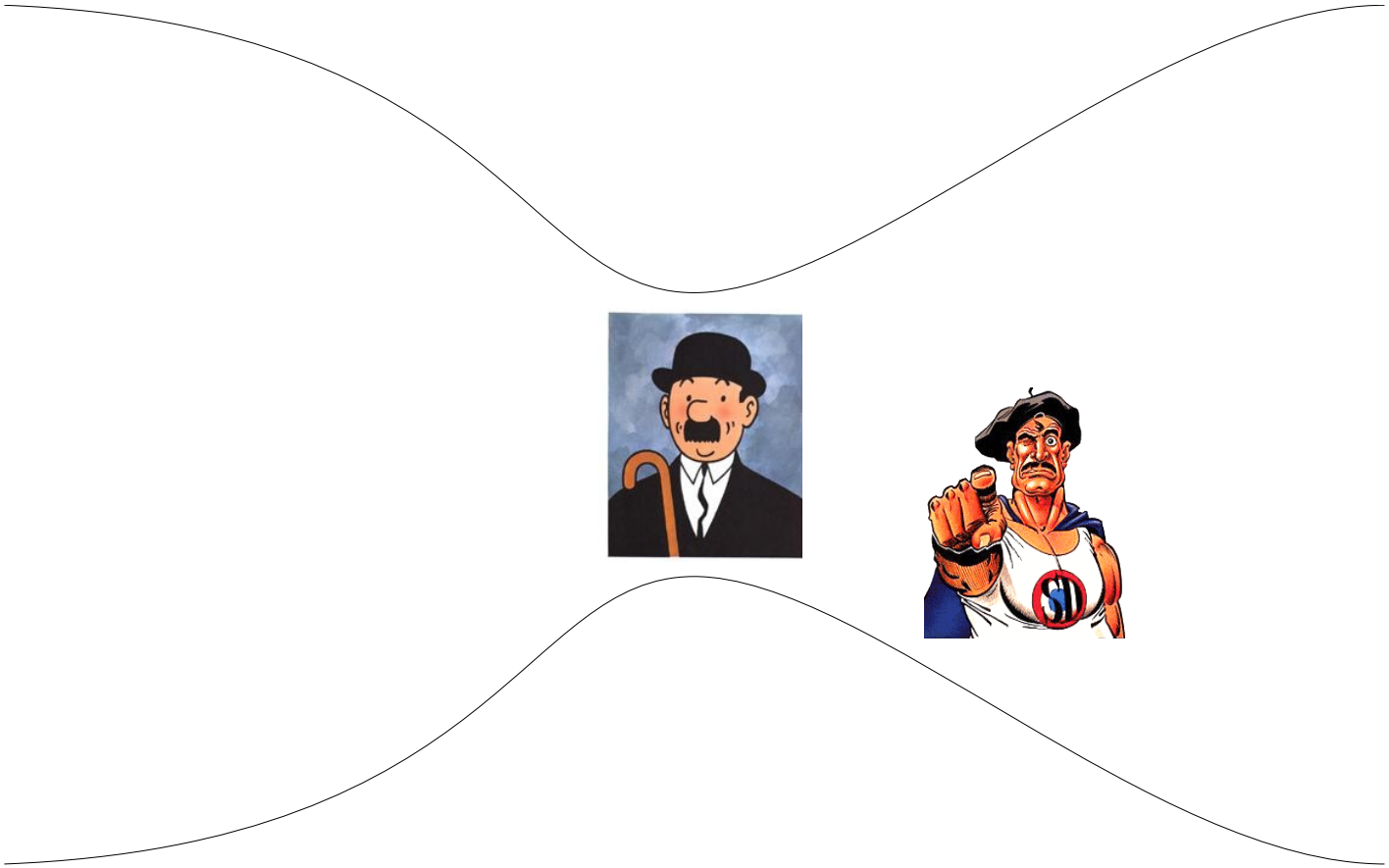
Exemple: avance parallèle

Exemple: demande d'accès à la ressource

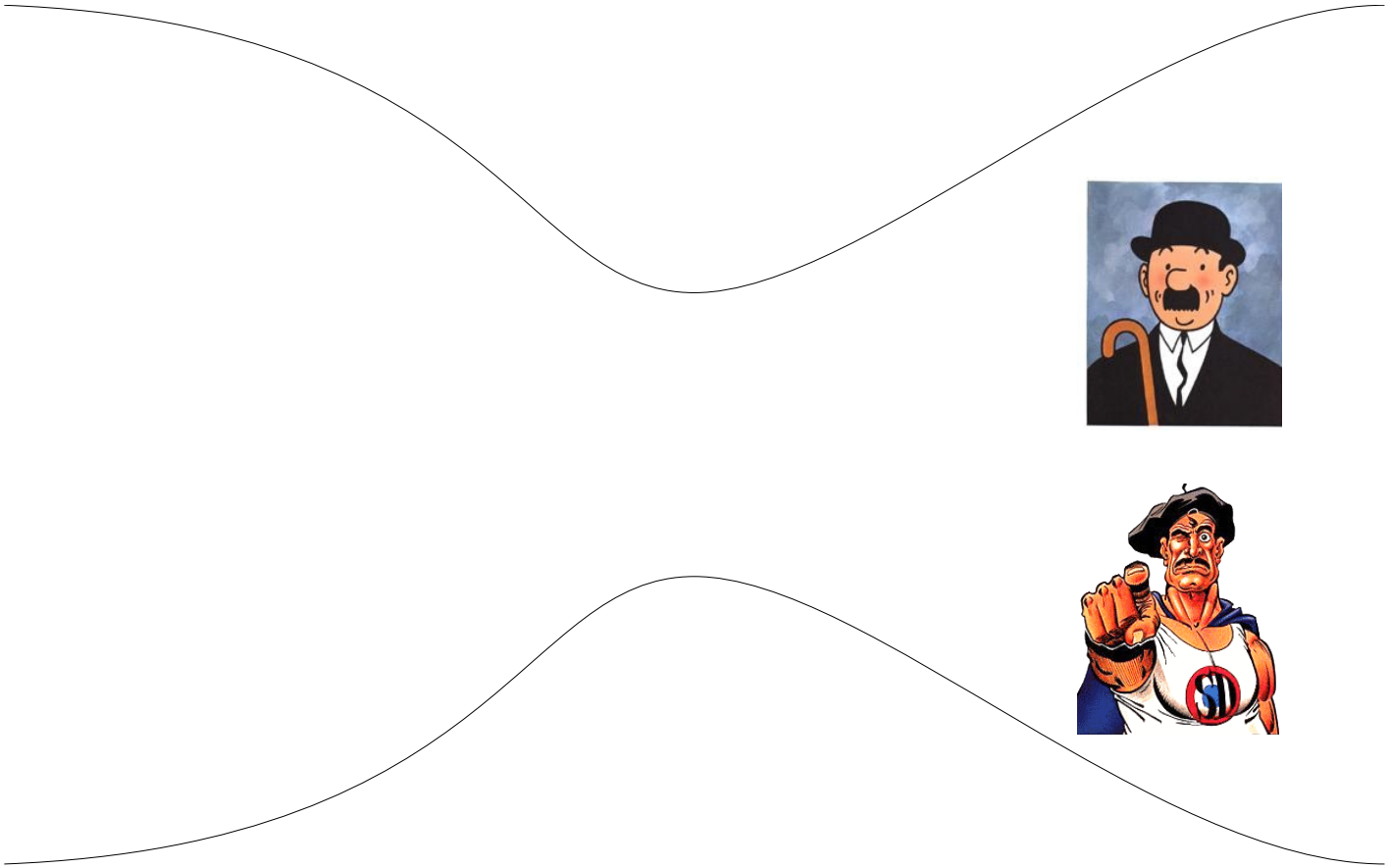
Exemple: accès à la ressource partagée

Exemple: relâchement de la ressource

Exemple: accès à la ressource partagée



Exemple: avance parallèle



Problème des threads: exemple

Classe GiveId

```
class GiveId {
private:
    int nb_id;
public:
    GiveId(): nb_id(0) {};
    int getUniqueId() {
        return nb_id ++;
    }
}
```

- Que signifie `return nb_id ++;` ?

```
int temp = nb_id;
nb_id = nb_id + 1;
return temp;
```

Problèmes des threads: exemple

```
int temp = nb_id;
nb_id = nb_id + 1;
return temp;
```

Condition initale:

```
nb_id = 0;
```

Exemple de déroulement

Thread A	Thread B
	
.	.
temp = 0	.
.	temp = 0
.	nb_id++
.	return 0
nb_id++	.
return 0	.
.	.

Implémentation

- Comment implémenter la concurrence?
  - Grâce à des mécanismes du langage de programmation
  - Grâce à des bibliothèques
  - Solution intermédiaire

Grâce au langage

- Avantages:
  - Les notions concurrentes de même que les constructions sont données par le langage
  - Détection d'une partie des erreurs à la compilation
  - Méthodologie de programmation imposée par le langage
- Désavantages :
  - Obligation d'utiliser un langage dédié qui est potentiellement peu répandu
  - Contraintes liées au langage choisi (pas forcément souhaitable)
- Exemples : ADA, Java, C++11, etc.

Grâce à des bibliothèques

- Le système d'exploitation offre une bibliothèque appelée *système multi-tâche*
- Avantage:
  - Un langage quelconque peut profiter de la bibliothèque
- Désavantages:
  - La portabilité (dépendance au système cible)
  - Déboguage délicat
  - Pas de méthodologie de programmation imposée
- Exemple: langage C ou C++ avec bibliothèque *POSIX Threads* aussi appelée *Pthreads*

Solution intermédiaire

- Un précompilateur gère l'implémentation des outils
- Exemple: C++ avec librairie Qt
  - Code indépendant de la plateforme cible

Exemple de langage: Ada

- Pour
  - Le langage intègre des notions de concurrence
  - Certaines vérifications peuvent être faites à la compilation
  - Robuste
- Contre
  - Moins utilisé que les autres en pratique

Exemple de langage: Java

- Pour
  - Orienté objet
  - Mécanismes de synchronisation natifs
    - Thread
    - synchronized
    - wait, notify, notifyAll
- Contre
  - Langage interprété
  - Donc: légèrement plus lent que C/C++
  - Définition un peu légère du fonctionnement des primitives

Exemple de langage: C/C++ (POSIX)

- Utilisation de la bibliothèque POSIX
- Pour
  - Très utilisé dans le monde (notamment embarqué)
  - Code compilé (rapidité)
- Contre
  - Pas de mécanismes de synchronisation natifs
  - Stroustrup: "It is possible to design concurrency support libraries that approach built-in concurrency support both in convenience and efficiency. By relying on libraries, you can support a variety of concurrency models, ..."[1]

[1] Parallel and Distributed Programming Using C++, Hughes et Hughes

Exemple de langage: C++11

- C++11 introduit des classes liées à la concurrence:
  - `thread`, `mutex`, `condition_variable`, `atomic`, `future`
  - Et des classes proches
- Pour
  - Contenu dans le langage
  - Lancement des threads facilité
- Contre
  - Pas de sémaphores
  - Il faut disposer d'un compilateur supportant C++11

Exemple de langage: C/C++ (Qt)

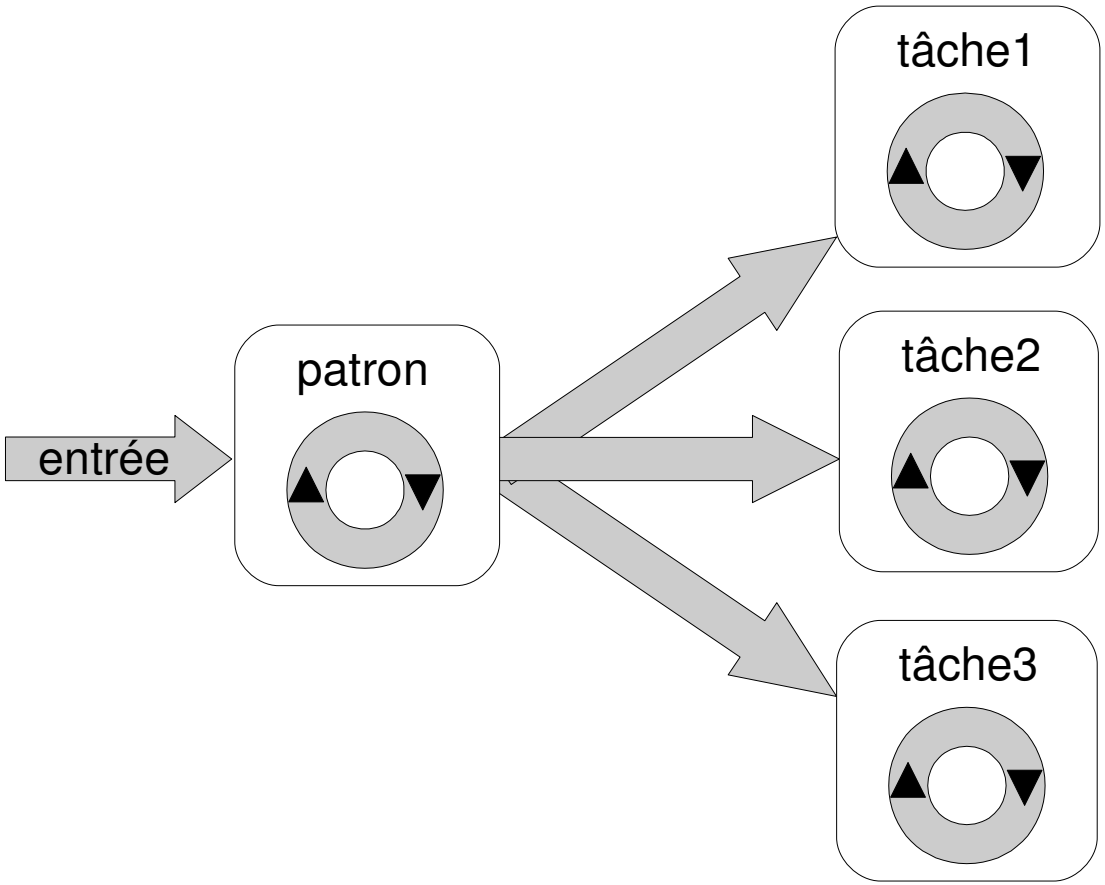
- Utilisation de l'environnement Qt
- Pour
  - Interopérabilité (Linux, Linux embarqué, Windows, Mac OS, Android, iOS)
  - Tous les objets nécessaires existent
    - **`QThread`**, **`QMutex`**, **`QSemaphore`**, **`QWaitCondition`**
  - Orienté objet
- Contre
  - Nécessite d'avoir un portage pour la plateforme
    - De plus en plus courant

Séparation d'un programme en threads

- Comment décomposer un programme en plusieurs threads?
- Il existe plusieurs modèles
  - Le modèle *délégation* (*boss-worker model* ou *delegation model* en anglais)
  - Le modèle *pair* (*peer model* en anglais)
  - Le modèle *pipeline* (*pipeline model* en anglais)

Modèle délégation

- Un thread principal
- Des threads travailleurs



Modèle délégation: exemple 1

```
void *patron(void *) {
  boucle infinie {
    attend une requête
    switch (requete) {
      case requeteX: startThread( ... tacheX); break;
      case requeteY: startThread( ... tacheY); break;
      ...
    }
  }
}

void *tacheX(void *) {
  exécuter le travail demandé, puis se terminer
}

void *tacheY(void *) {
  exécuter le travail demandé, puis se terminer
}
```

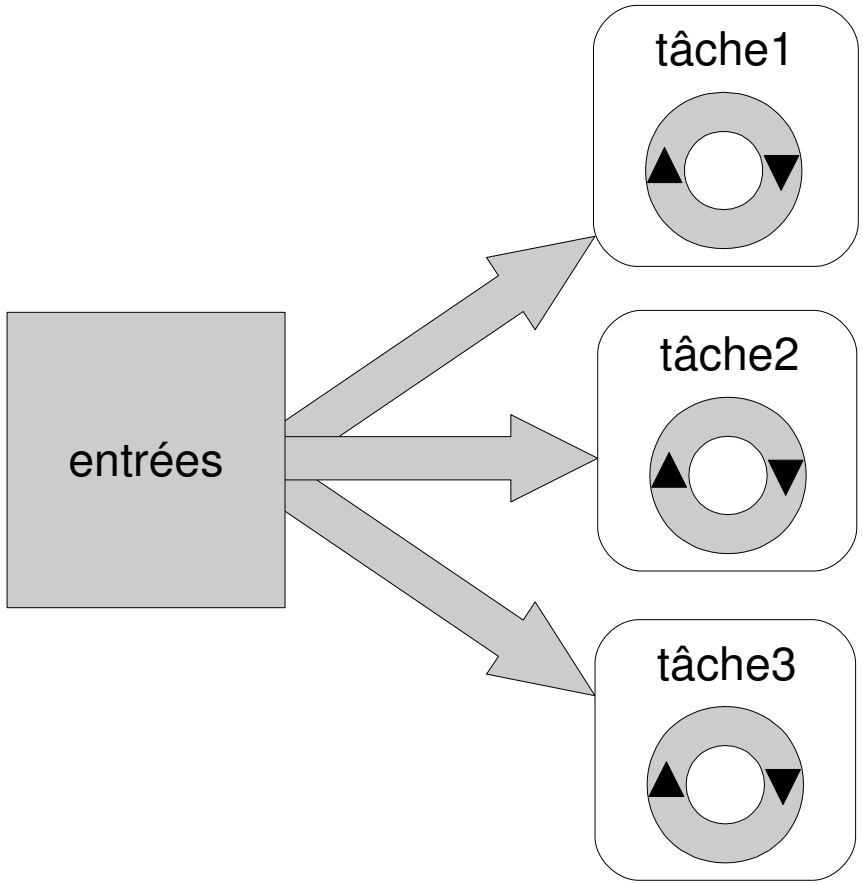
Modèle délégation: exemple 2

```
void *patron(void *) {
  // crée tous les threads
  startThread(...);
  boucle infinie {
    attend une requête;
    place la requête dans la file d'attente
    signale aux travailleurs qu'une requête est prête
  }
}

void *travailleur(void *) {
  boucle infinie {
    bloque jusqu'à être activé par le patron
    récupère la requête de la file d'attente
    switch(requete){
      case requeteX: tacheX();
      case requeteY: tacheY();
      ...
    }
```

Modèle pair

- Pas de thread principal
- Tous égaux
- Chacun s'arrange avec ses entrées/sorties



Modèle pair: exemple

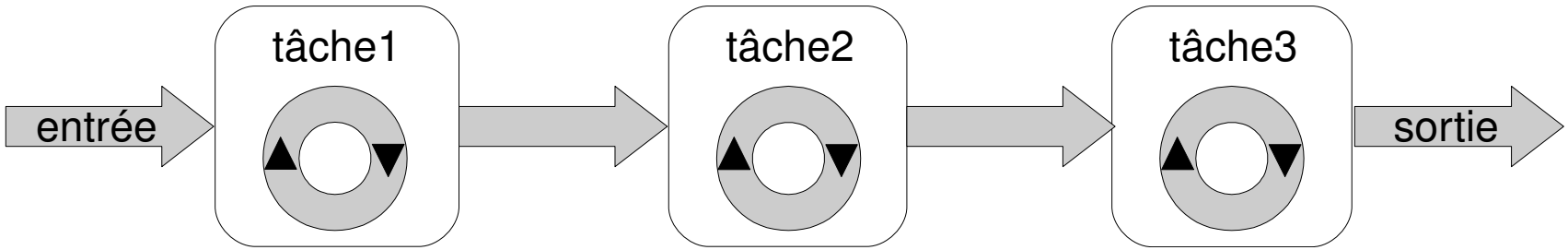
```
main() {
  startThread( ... tache1);
  startThread( ... tache2);
  ...
  signale aux threads qu'ils peuvent commencer à travailler
}

tache1() {
  attend le signal de commencement
  effectue le traitement, et synchronise avec les autres threads
  si nécessaire
}

tache2() {
  attend le signal de commencement
  effectue le traitement, et synchronise avec les autres threads
  si nécessaire
}
```

Modèle pipeline

- Appliqué lorsque:
  - L'application traite une longue chaîne d'entrée;
  - Le traitement à effectuer sur ces entrée peut être décomposé en sous-tâches (étages de pipeline) au travers desquelles chaque donnée d'entrée doit passer;
  - Chaque étage peut traiter une donnée différente à chaque instant.
- Un thread attend les données du précédent
- Et les transmet ensuite au suivant



Modèle pipeline: exemple (1)

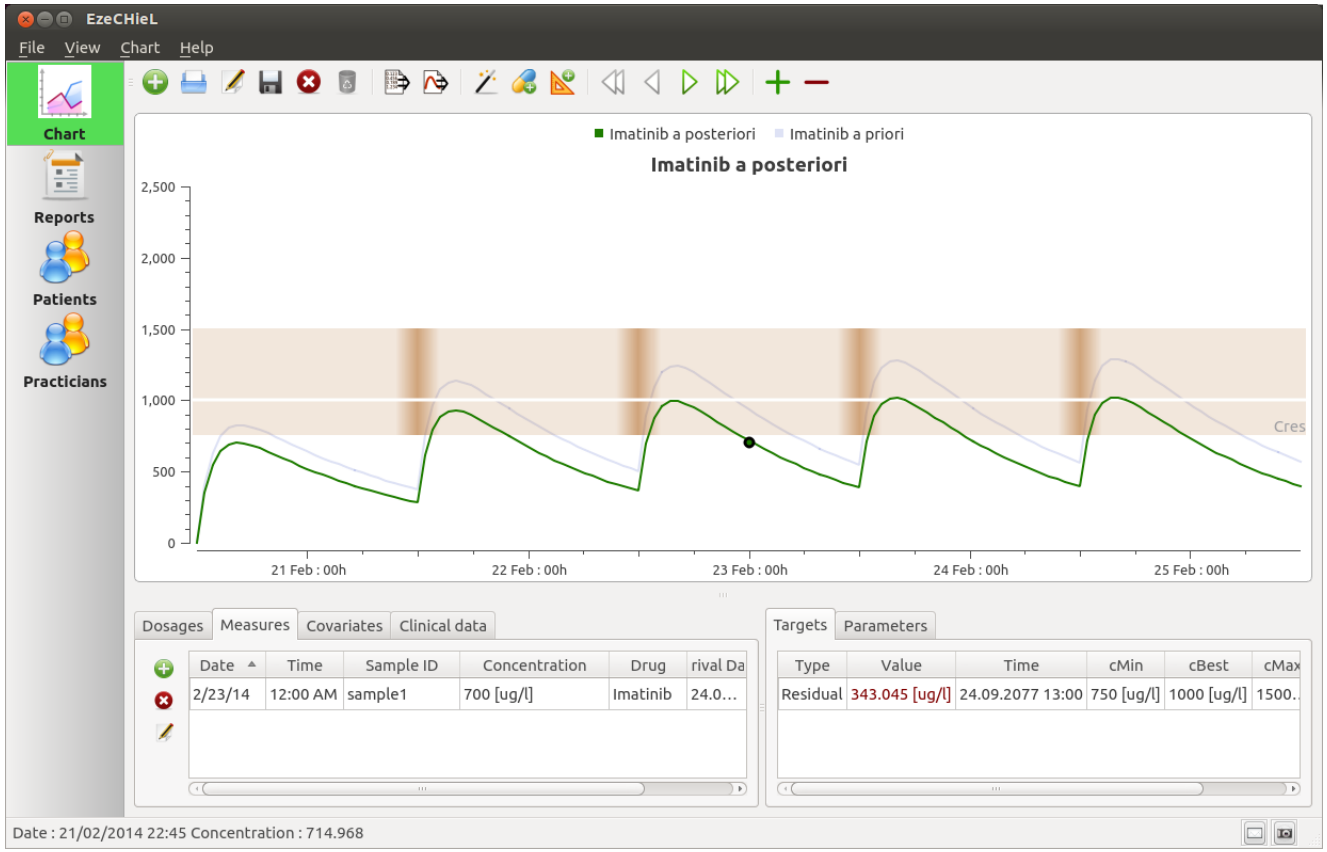
```
etage1() {
  boucle infinie {
    récupérer une entrée du programme
    traiter cette donnée
    passer le résultat à l'étage suivant
  }
}
etage2() {
  boucle infinie {
    récupérer une donnée de l'étage précédent
    traiter cette donnée
    passer le résultat à l'étage suivant
  }
}
etageN() {
  boucle infinie {
    récupérer une donnée de l'étage précédent
    traiter cette donnée
    passer le résultat en sortie du programme
```

Modèle pipeline: exemple (2)

```
main() {
  startThread( ... etage1);
  startThread( ... etage2);
  ...
  startThread( ... etageN);
  ...
}
```

Exemple d'EzeCHieL

- Logiciel d'aide à l'interprétation de mesures de concentration pour ajustement de posologies (pour pharmacologie clinique)

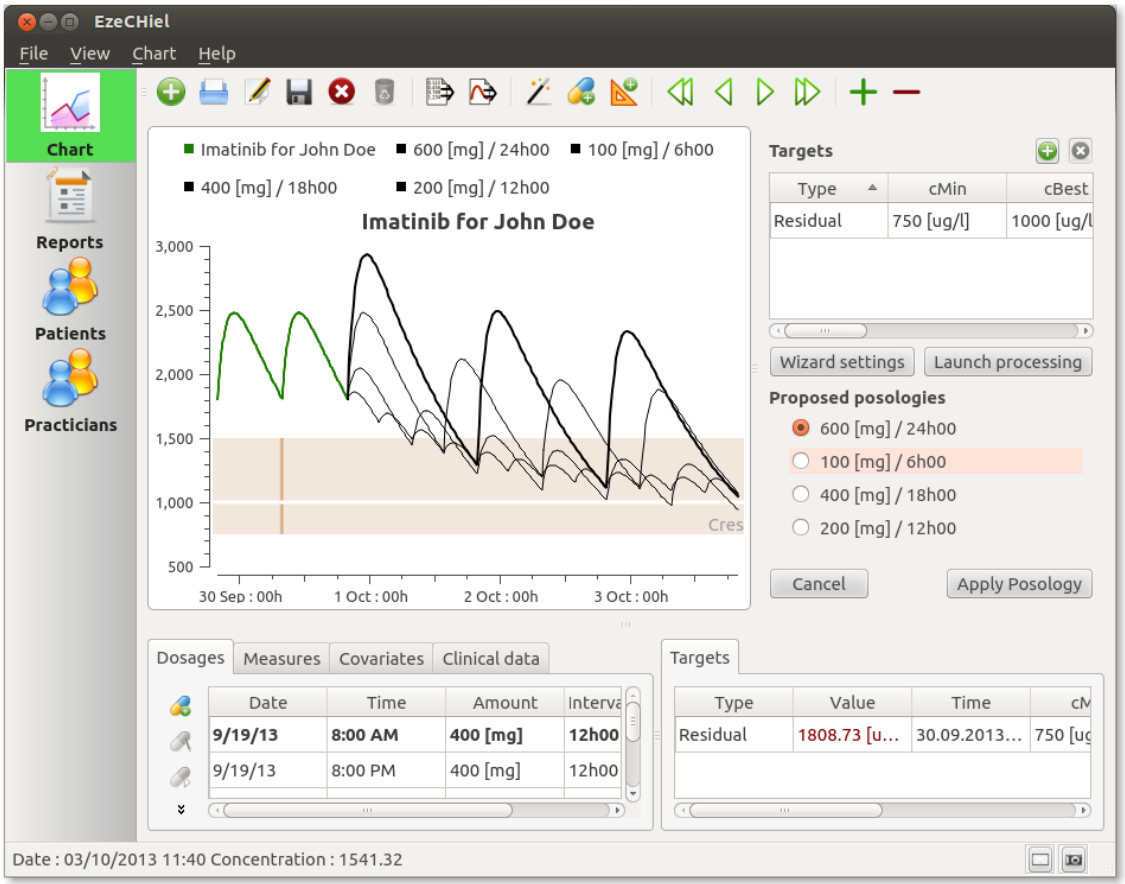


Exemples sur le projet EzeCHieL

- Les médicaments sont décrits par des fichiers XML
- Ils doivent être chargés au lancement du programme
  - ⇒ Chargement géré par un thread
    - Evite de ralentir le lancement du programme

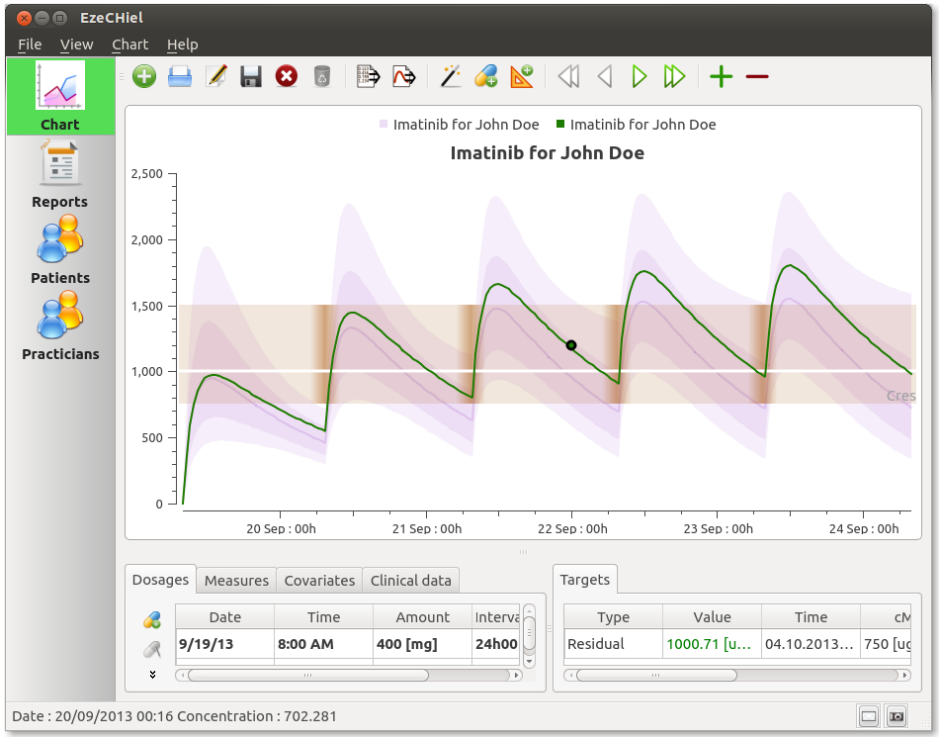
Exemples sur le projet EzeCHieL

- Certains traitements mathématiques nécessaires pour afficher une courbe sont longs (quelques secondes)
  - ⇒ Traitement géré par un thread
    - Evite de freezer l'application pendant le traitement



Exemples sur le projet EzeCHieL

- Calcul de courbes de percentiles selon Monte Carlo
- Il faut calculer 3000 courbes
  - ⇒ Multithreadé
- Il faut trier ces courbes, à chaque temps
  - ⇒ Multithreadé



Code source