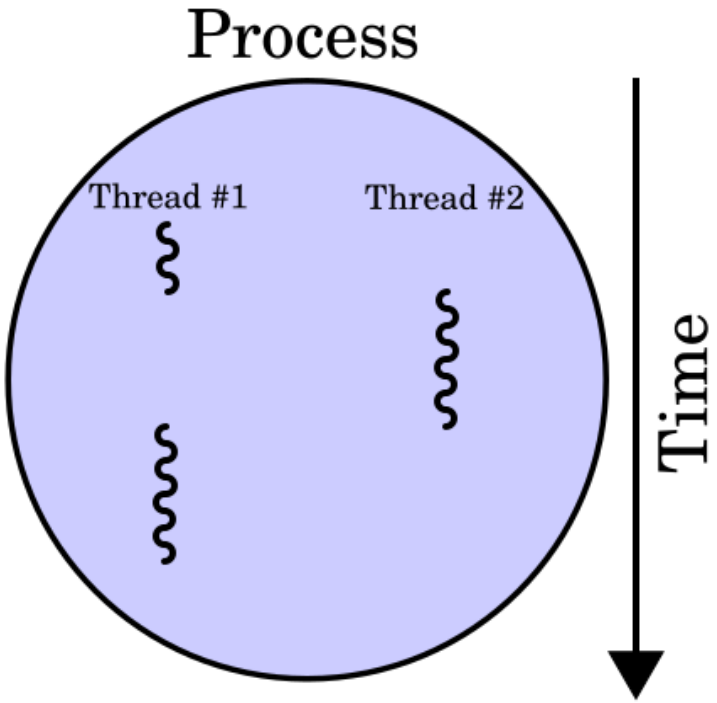
Threads

Les threads

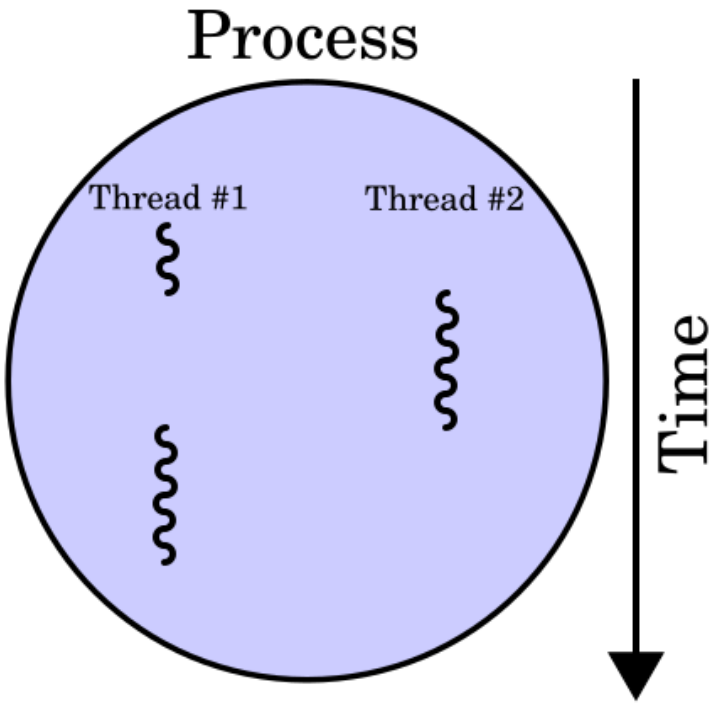
- Jusqu'à présent, dans un processus on n'a qu'un seul chemin (ou fil) d'exécution



Un processus avec deux *threads*.

Les threads

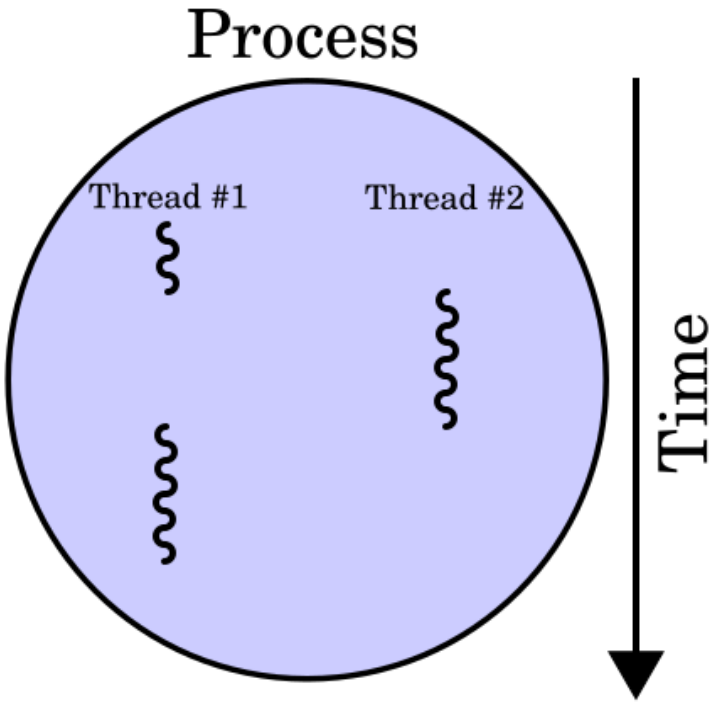
- Jusqu'à présent, dans un processus on n'a qu'un seul chemin (ou fil) d'exécution
- Avec les *threads*, il est possible d'avoir plusieurs fils qui s'exécutent simultanément



Un processus avec deux *threads*.

Les threads

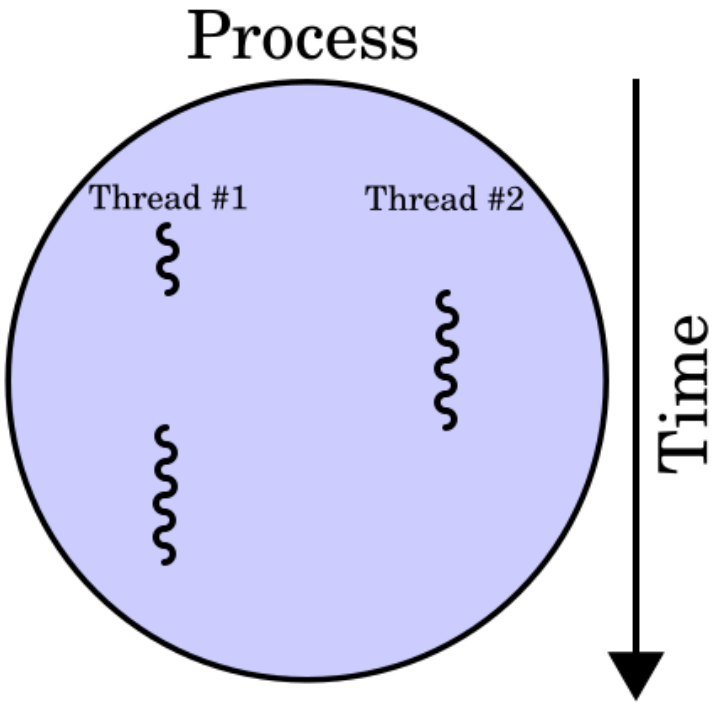
- Jusqu'à présent, dans un processus on n'a qu'un seul chemin (ou fil) d'exécution
- Avec les *threads*, il est possible d'avoir plusieurs fils qui s'exécutent simultanément
- Les différents threads d'un processus partagent les mêmes données



Un processus avec deux *threads*.

Les threads

- Jusqu'à présent, dans un processus on n'a qu'un seul chemin (ou fil) d'exécution
- Avec les *threads*, il est possible d'avoir plusieurs fils qui s'exécutent simultanément
- Les différents threads d'un processus partagent les mêmes données
- Si plusieurs threads utilisent les mêmes données il faut en général mettre en place des mécanismes d'exclusion mutuelle



Un processus avec deux *threads*.

Pourquoi les threads ?

- Interfaces graphiques
  - les interactions avec l'utilisateur et les calculs sont gérés par des threads différents

Pourquoi les threads ?

- Interfaces graphiques
  - les interactions avec l'utilisateur et les calculs sont gérés par des threads différents
- Calcul intensif
  - parallélisation des calculs pour utiliser plusieurs processeurs

Pourquoi les threads ?

- Interfaces graphiques
  - les interactions avec l'utilisateur et les calculs sont gérés par des threads différents
- Calcul intensif
  - parallélisation des calculs pour utiliser plusieurs processeurs
- Télécommunications
  - communications asynchrones

Pourquoi les threads ?

- Interfaces graphiques
  - les interactions avec l'utilisateur et les calculs sont gérés par des threads différents
- Calcul intensif
  - parallélisation des calculs pour utiliser plusieurs processeurs
- Télécommunications
  - communications asynchrones
- ...

Mise en œuvre C

- Bibliothèque *pthreads* (*POSIX threads)*

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
void pthread_exit(void *retval);
int pthread_join(pthread_t thread, void **retval);
```

- Faire la compilation et l'édition de liens avec l'option `-pthread`

Mise en œuvre en C : *pthreads*

- Un thread est créé et démarré par un appel à `pthread_create()`

Mise en œuvre en C : *pthreads*

- Un thread est créé et démarré par un appel à `pthread_create()`
- Le thread démarre en exécutant la fonction `start_routine()` ; `arg` est passé en paramètre à `start_routine()`

Mise en œuvre en C : *pthreads*

- Un thread est créé et démarré par un appel à `pthread_create()`
- Le thread démarre en exécutant la fonction `start_routine()` ; `arg` est passé en paramètre à `start_routine()`
- Le paramètre `attr` permet de préciser des options pour la création du thread (on utilisera NULL pour les options par défaut)

Mise en œuvre en C : *pthreads*

- Un thread est créé et démarré par un appel à `pthread_create()`
- Le thread démarre en exécutant la fonction `start_routine()` ; `arg` est passé en paramètre à `start_routine()`
- Le paramètre `attr` permet de préciser des options pour la création du thread (on utilisera NULL pour les options par défaut)
- Le thread se termine par un retour de la fonction `start_routine()` ou par un appel à `pthread_exit()`

Mise en œuvre en C : *pthreads*

- Un thread est créé et démarré par un appel à `pthread_create()`
- Le thread démarre en exécutant la fonction `start_routine()` ; `arg` est passé en paramètre à `start_routine()`
- Le paramètre `attr` permet de préciser des options pour la création du thread (on utilisera NULL pour les options par défaut)
- Le thread se termine par un retour de la fonction `start_routine()` ou par un appel à `pthread_exit()`
- L'identifiant du thread est stocké dans la variable pointée par `thread`

Mise en œuvre en C : *pthreads*

- Un thread est créé et démarré par un appel à `pthread_create()`
- Le thread démarre en exécutant la fonction `start_routine()` ; `arg` est passé en paramètre à `start_routine()`
- Le paramètre `attr` permet de préciser des options pour la création du thread (on utilisera NULL pour les options par défaut)
- Le thread se termine par un retour de la fonction `start_routine()` ou par un appel à `pthread_exit()`
- L'identifiant du thread est stocké dans la variable pointée par `thread`
- On peut attendre la fin d'un thread et récupérer sa valeur de retour avec `pthread_join()`

Exemple en C

```c
#include <pthread.h>
#include <stdio.h>

void *ma_jolie_fonction(void *arg __attribute__((unused)))
{
    printf("Coucou, je suis le thread !\n");
    return NULL;
}

int main(void)
{
    pthread_t th;
    pthread_create(&th, NULL, ma_jolie_fonction, NULL);
    printf("Hop !\n");
    pthread_join(th, NULL);
    return 0;
}
```

Mise en œuvre en Java

- Déclaration :

Mise en œuvre en Java

- Déclaration :
  - Dériver de la classe `Thread`

Mise en œuvre en Java

- Déclaration :
  - Dériver de la classe `Thread`
  - Implémenter la méthode `public void run()` avec le code du thread

Mise en œuvre en Java

- Déclaration :
  - Dériver de la classe `Thread`
  - Implémenter la méthode `public void run()` avec le code du thread
- Utilisation :

Mise en œuvre en Java

- Déclaration :
  - Dériver de la classe `Thread`
  - Implémenter la méthode `public void run()` avec le code du thread
- Utilisation :
  - Construire un objet de ladite classe

Mise en œuvre en Java

- Déclaration :
  - Dériver de la classe `Thread`
  - Implémenter la méthode `public void run()` avec le code du thread
- Utilisation :
  - Construire un objet de ladite classe
  - Démarrer le thread avec la méthode `start()`

Mise en œuvre en Java

- Déclaration :
  - Dériver de la classe `Thread`
  - Implémenter la méthode `public void run()` avec le code du thread
- Utilisation :
  - Construire un objet de ladite classe
  - Démarrer le thread avec la méthode `start()`
  - Attendre la fin d'un thread avec la méthode `join()`

Exemple en Java

```
class Exemple extends Thread {

    public void run() {
        System.out.println("Coucou, je suis le thread !");
    }

    public static void main(String[] args) {
        Exemple th = new Exemple();
        th.start();
        System.out.println("Hop !");
        try {
            th.join();
        } catch (InterruptedException e) { }
    }
}
```