MODULE 107
MULTI-THREADING

PLAN

- Introduction
- Passage de messages
- Partage de données
- Parallélisme des données en Rust

INTRODUCTION

RUST GUARANTEES

Rust garantie:

- a sûreté de l'accès à la mémoire entre les threads.

Rust ne garantie pas:

- a logique du programme,
- l'absence de deadlocks, livelocks, etc.

MODÈLE DE THREADING

- Librairie standard~: 1 thread Rust = 1 thread système (OS)
- Pas de runtime, abstraction à 0 coût
- D'autres librairies pour d'autres modèles (ex. green threads)

LANCER DES THREADS 1/2

```rust
use std::thread;
thread::spawn(|| { // 👉 closure, may capture some context
    println!("Hello from thread!");
});
```

LANCER DES THREADS 2/2

```rust
use std::thread;
let handle = thread::spawn(|| {
    println!("Hello from thread!");
});

// 👇 handle to manipulate the created thread
handle.join().unwrap();
```

EXERCICE

EXERCICE 1 - LANCER DES THREADS

PASSAGE DE MESSAGES

PASSAGE DE MESSAGES 1/3

- Une des manières la plus simple de communiquer entre les threads
- Voir std::sync::mpsc::channel
  - A un seul thread receveur,
  - A un ou plusieurs émetteurs,
  - Permet de passer des messages d'un seul type,
  - Agit comme une file pour le receveur.

PASSAGE DE MESSAGES 2/3

Le receveur peut

- Récupérer immédiatement le prochain message,
- Bloquer en attendant le prochain message,
- Avec un timeout, ou une date butoir (deadline).

PASSAGE DE MESSAGES 3/3

```rust
use std::thread;
use std::sync::mpsc;

fn main() {
    let (tx, rx) = mpsc::channel();
    thread::spawn(move || {
        let val = String::from("hi");
        tx.send(val).unwrap();
    });
    let received = rx.recv().unwrap();
    println!("Got: {}", received);
}
```

EXERCICE

EXERCICE 2 - PASSAGE DE MESSAGES

PARTAGE DE DONNÉES

PARTAGE DE DONNÉES - STD

Dans la librairie standard: sync module

- Arc: Atomically Reference Counted, référence partagée dans un context multi-thread
- Mutex: Encapsule une ressource pour un accès exclusif
- RwLock: Similaire à `Mutex` mais autorise plusieurs lecteurs
- Barrier: Mise en attente de plusieurs threads en attente d'un évènement

VARIABLES ATOMIQUES

Dans la librairie standard: atomic module

- Structures de la forme `AtomicI32` pour les versions atomiques des types primitifs.
- Rust expose le modèle de mémoire de C++20 (à travers LLVM)
- Les APIs utilisent un Ordering (similaire à C++20).

SEND ET SYNC

- Send marque les types dont la propriété (ownership) peut être transféré d'un thread à l'autre.
- Sync marque les types dont les références à ceux-ci peuvent être partagées entre plusieurs threads.
  - si T est Sync, &T peut être partagé entre threads.

SEND ET SYNC - DÉRIVATION

- Send et Sync sont automatiquement dérivés par le compilateur
- Les types primitifs sont quasiment tous marqués Send et Sync.
- Les types dont les membres sont tous Send est automatiquement marqué avec Send.
- Idem pour Sync.

SEND ET SYNC - EXEMPLES

Exemples de types qui ne sont pas Send ou Sync:

- Cell, RefCell, Rc
- pointeurs `*const T` and `*mut T`

SEND ET SYNC - IMPLÉMENTATION MANUELLE

Toujours `unsafe`!

```
struct MyBox(*mut u8);

unsafe impl Send for MyBox {}
unsafe impl Sync for MyBox {}
```

Dans ce cas, il appartient au programmeur de s'assurer que l'usage de ce type respecte les garantie de sûreté de Rust.

SEND ET SYNC - IMPLÉMENTATION NÉGATIVE

Empêcher la dérivation automatique:

```
#![feature(optin_builtin_traits)]

// I have some magic semantics
// for some synchronization primitive!
struct SpecialThreadToken(u8);

impl !Send for SpecialThreadToken {}
impl !Sync for SpecialThreadToken {}
```

EXERCICE

EXERCICE 3 - PARTAGE DE DONNÉES

PARALLÉLISME DES DONNÉES EN RUST

RAYON

- rayon est une lib. pour le parallélisme des données
- fournit des constructions de haut niveau
  - itérateurs parallèles
  - pool de threads save work-stealing
- facile à utiliser et sûr!

RAYON - EXEMPLE

```
let inputs = vec![/* -- snip -- */];
let outputs: Vec<T> = inputs
                .iter()
                .map(|item| do_stuff(item))
                .collect();
```

RAYON - EXEMPLE

```rust
let inputs = vec![/* -- snip -- */];
let outputs: Vec<T> = inputs
                .par_iter() // 👉 here
                .map(|item| do_stuff(item))
                .collect();
```

EXERCICE

EXERCICE 4 - RAYON