Travail pratique de programmation système avancée

Préparation de petit déjeuner

1 Objectifs

- Modélisation asynchrone
- Familiarisation avec la syntaxe `async/await`

2 Énoncé

2.1 Préparation du petit déjeuner

La préparation d'un petit déjeuner consiste en différentes étapes qu'on peut considérer comme étant exécutée séquentiellement:

1. Mettre le pain dans le grille-pain pour le faire chauffer (90s)
2. Beurrer le pain (10s)
3. Mettre la confiture sur la tartine (10s)
4. Chauffer l'eau avec la bouilloire (60s)
5. Mettre le sachet de thé dans la tasse (10s)
6. Remplir la tasse avec l'eau chaude (10s)
7. Mettre le lait dans la tasse (10s)
8. Attendre un peu que le thé refroidisse (30s)

Une personne qui ferait ces tâches dans l'ordre mettrait donc 230 secondes à préparer un thé et une tartine. En supposant que les transitions et le service sont plus ou moins instantannées.

Orestis a 3 enfants, il lui faudrait donc 690 secondes pour préparer le petit déjeuner de tout le monde et le matin il a pas que ça à faire. On aimerait donc l'aider à accélérer ce processus. En effet, certaines parties de notre tâche principale sont des attentes qui peuvent être mises à profit pour effectuer d'autres tâches pendant l'attente.

Il se peut que les enfants d'Orestis invitent plein d'amis à dormir. Il faudrait donc un moyen de mesurer le temps que ça prendrait pour faire le petit déjeuner pour `N` enfants.

Pour ce faire il faudra utiliser la fonction `join_all()` de la librairie `futures-util`, voir ce lien pour la doc.

2.2 Modélisation

Séparer la tâche (en 7 parties) ci-dessus en 3 sous-tâches qui regroupent certains items. L'idée principale est d'avoir une modélisation qui permette de rendre notre tâche asynchrone et ainsi l'accélérer. Modélisez les actions qui peuvent être non-bloquantes.

2.3 Code

Écrire un programme en Rust permettant de mesurer le gain de temps entre une personne seule effectuant les tâches de façon bloquante, et une de façon asynchrone (quand cela s'y prête). Pour vous assurer d'avoir un seul thread, vous pouvez utiliser:

```
#[tokio::main(flavor = "multi_thread", worker_threads = 1)]
```

2.4 Indications

Il y a deux fonctions qui permettent d'attendre en qui sont intéressantes pour nous:

```
use tokio::time::sleep; // non-bloquant
use std::thread::sleep; // bloquant
```

Pour les attentes, utilisez les temps ci-dessus, mais en milisecondes pour pas attendre des heures...