Travail pratique de programmation système avancée

Le runtime un peu plus générique

1 Objectifs

- Création d'un runtime
- Généricité dans l'appel des fonctions

2 Énoncé

2.1 Généricité

Le but de ce travail pratique est de "copier" l'API de la librairie standard de `std::thread::spawn()`. Ainsi l'appel à `spawn()` pourrait avoir l'air de

```
Runtime::spawn(move || {
    // do some cool stuff with the environement
});
```

Pour ce faire, il faudra faire plusieurs modification au code que nous avons construit en cours.

1. Lors de l'appel à `spawn()` on passe en argument une fonction de type `fn()` avec la fonction

   ```
   fn spawn<F: FnOnce() + 'static>(f: F);
   ```

   où on voit que la fonction `fn()` est remplacée par n'importe quelle fermeture (closure) dont la durée de vie est statique (a la durée de vie du programme).

2. L'appel à `spawn()` ne prend plus `self` en argument

3. Il faut ajouter un thread `thread_ptr` à la struct `ThreadContext` qui va permettre d'enregistrer l'adresse de la fonction passée en argument et permettre son exécution.

4. Il faut ajouter un membre `task` de type `Option<Box<dyn FnOnce>>` au type `Thread`, ansi qu'un `id: usize` pour identifier les threads.

5. Il faut remplacer la fonction `f()` dans l'appel à `write()` par l'appel à une fonction `call(thread: u64)` qui à partir du `thread_ptr` va appeler la bonne tâche.

6. La fonction `switch` doit prendre en compte l'ajout de `tread_ptr` après le dernier `mov`

7. Pour bien voir quand les threads se terminent, modifiez la fonction `guard()` pour qu'on affiche l'`id` du thread.
8. Modifier également la fonction `run()` pour qu'elle ne prenne plus `self` en argument.

Le `main()` de votre programme peut ressembler à cela:

```
fn main() {
    let mut runtime = Runtime::new();
    runtime.init();
    for num in 0..MAX_THREADS - 2 {
        Runtime::spawn(move || {
            println!("Starting thread {}", num + 1);
            for i in 0..10 {
                println!("Thread num {num}, counter {i}");
                yield_thread();
            }

            Runtime::spawn(move || {
                println!("Spawned thread num {num}");
            });
        });
    }
    Runtime::run();
}
```

2.2 Exercice supplémentaire

Rendre le scheduler un peu plus équitable. Choisir le prochain thread à exécuter en fonction du temps que chaque thread a déjà passé à être exécuté. De tous les threads prêts prendre celui qui a été exécuté le moins longtemps depuis le début de l'exécution.