Dining Philosopher Problem

(Github Repo) (Last push to the Rust source code 10/18/2022)

CECS 326 - Spring 2022

Stephen Lyons, 025785875

Noah Daniels, 026466906

Professor Hailu Xu

California State University, Long Beach

College of Engineering

1250 Bellflower Blvd, Long Beach, CA 90840

Due 10/15/2022

Project Architecture

This project was programmed in the Rust programming language and the rust toolchain was used to compile and run the code. All classes and the main code are written in the main.rs file. The code can be run via the executable file or by installing the rust toolchain:

To install Rust

1. Install Rustup: <https://rustup.rs/>
2. Install Rust stable: `rustup install stable`

To run the project

1. `cd` into the project directory `project2rust`
2. `cargo run`
or
3. Run the executable in the target directory (project2rust\target\debug\project2rust.exe)

The Philosopher struct (class) has four fields: name, left_fork, right_fork, and eat_count. The name field is a string that holds the name of the philosopher. The left_fork and right_fork fields are mutexes that represent the forks. The Philosopher struct has two methods: eat and think. The eat method locks the left and right forks and then prints a message that the philosopher is eating. The think method unlocks the left and right forks and then prints a message that the philosopher is thinking. Both the thinking and eating functions take a random amount of time to simulate the philosopher thinking or eating, this is done by using the rand crate. Once the Philosopher is finished eating or thinking, it calls the other respective function

```rust
// define the function implementations for the philosopher struct
impl Philosopher {
    // constructor for the philosopher struct
    fn new(name: &str, left: usize, right: usize, index: usize) -> Philosopher {
        Philosopher {
            name: name.to_string(),
            left: left,
            right: right,
            index: index,
        }
    }
    // function to eat which will call the take_forks and return_forks functions
    pub fn eat(&self, table: &DiningServer) {
        // sleep for a random period between one and three seconds
        let sleep_time = rand::thread_rng().gen_range(0..4);
        thread::sleep(Duration::from_secs(sleep_time));

        table.take_forks(self);
        // return forks after eating so other philosophers can eat
        table.return_forks(self);
        println!(
            "Philosopher #{} took {} seconds to eat",
            self.index, sleep_time
        );
        // switch the state of the philosopher to thinking
        self.think(&table);
    }
    // function to think which will only wait for a random period between one and three seconds and print
    pub fn think(&self, table: &DiningServer) {
        // sleep for a random period between one and three seconds
        let sleep_time = rand::thread_rng().gen_range(1..4);
        thread::sleep(Duration::from_secs(sleep_time));
        println!(
            "Philosopher #{} took {} seconds to think",
            self.index, sleep_time
        );
        // switch the state of the philosopher to eating
        self.eat(&table);
    }
}
```

The Dining Server struct (class) has two fields: philosophers and forks. The philosophers field is a vector of Philosopher structs. The forks field is a vector of mutexes. The Dining Server implementation has two methods: take_forks and return_forks. The take_forks method takes a reference to a philosopher and locks the left and right forks. The return_forks method takes a reference to a philosopher and unlocks the left and right forks.

```rust
// function to take forks which will lock the mutexes for the left and right forks
fn take_forks(&self, philosopher: &Philosopher) {
    // bitwise AND operator to check if the index of the philosopher is even or odd
    if philosopher.index & 1 == 0 {
        // if the index is even, lock the left fork first
        let _left = self.forks[philosopher.left].lock().unwrap();
        // then lock the right fork
        thread::sleep(Duration::from_secs(1));
        let _right = self.forks[philosopher.right].lock().unwrap();
        println!(
            "Fork #{} is locked by Philosopher #{}",
            philosopher.left, philosopher.index
        );
        println!(
            "Fork #{} is locked by Philosopher #{}",
            philosopher.right, philosopher.index
        );
        println!("Philosopher #{} took forks", philosopher.index);
    } else {
        // if the index is odd, lock the right fork first
        let _right = self.forks[philosopher.right].lock().unwrap();
        thread::sleep(Duration::from_secs(1));
        let _left = self.forks[philosopher.left].lock().unwrap();
        println!(
            "Fork #{} is locked by Philosopher #{}",
            philosopher.right, philosopher.index
        );
        println!(
            "Fork #{} is locked by Philosopher #{}",
            philosopher.left, philosopher.index
        );
        println!("Philosopher #{} took forks", philosopher.index);
    }
}
// function to return forks which will unlock the mutexes for the left and right forks
fn return_forks(&self, philosopher: &Philosopher) {
    drop(self.forks[philosopher.left].lock().unwrap());
    drop(self.forks[philosopher.right].lock().unwrap());
}
```

```rust
// function to return forks which will unlock the mutexes for the left and right forks
fn return_forks(&self, philosopher: &Philosopher) {
    drop(self.forks[philosopher.left].lock().unwrap());
    drop(self.forks[philosopher.right].lock().unwrap());
}
```

The main function creates a new DiningServer, a vector of philosophers and forks and then spawns a thread for each philosopher. The main function then joins all the threads and prints the action of each philosopher and when they are done eating.

##Contributions

Stephen wrote the code for the project in Rust and Noahcontributed some of the supporting comments and portions of the main function. The report was written by both of us and the code execution video was recorded by Noah.