Parallelism/Concurrency in Rust

Lecture 11

Slides: Neil Kaushikkar

Goals For Today

- Motivation Behind Threads
- How to Spawn Threads in Rust
- Thread Joining

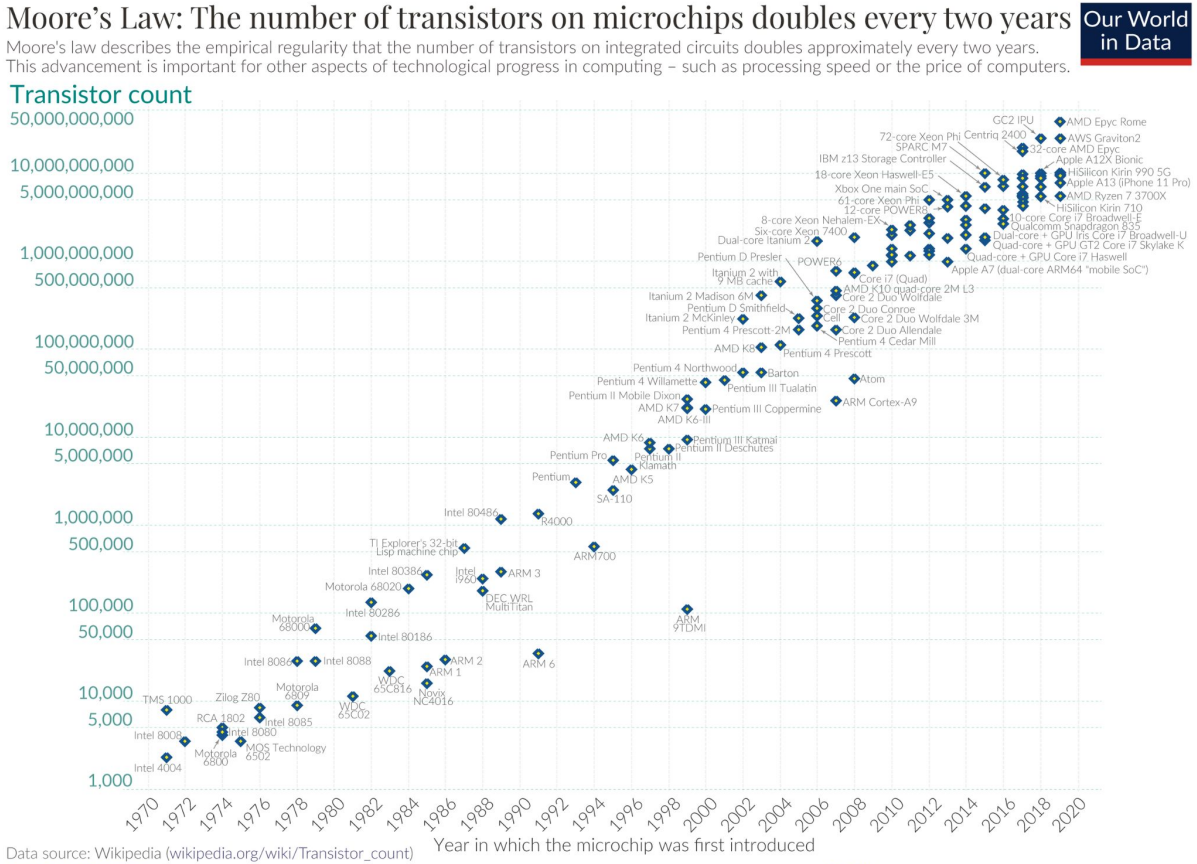
Reminders

- HW8 due 10/6 at 11:59pm
- HW9 releases today, due 10/11 at 11:59pm
- MP2 due date extended, due 10/11 at 11:59
    - 70% credit due date also pushed back

Programs and Processes

- What happens when you run your Rust program?
- A process is an instance of a program running in a computer
- In UNIX and some other operating systems, a process is started when a program is initiated
- How can we make programs (processes) faster?
    - Bigger CPUs?
    - Smaller transistors on a CPU?
    - More transistors on a CPU?

Moore’s Law



Reference:

- https://en.wikipedia.org/wiki/Moore%27s_law

Enter Multi-Core Machines

- Processor ≈ Core ≈ CPU → for the purposes of this lecture…
- A multicore processor is an integrated circuit that has two or more processor cores attached for enhanced performance
- These processors also enable more efficient simultaneous processing of multiple tasks, such as with **parallel processing** and **multithreading**
- Each processor runs programs (processes) separately
    - We can run multiple processes at the same time
    - How does this help speed up a single process?

Enter Threads

- A thread is short for ‘thread-of-execution’
- Represents the sequence of instructions that the CPU has and will execute
- Useful when you want to leverage the power of multi-core systems to do 1 task
    - Each thread has a ‘main method’
    - Run 1 thread per core
    - Run more code in the same amount of time (as 1 core)
- Threads allow for parallel and concurrent code execution

Reference:

- https://cs341.cs.illinois.edu/coursebook/Threads

Parallelism & Concurrency

- Concurrency: multiple computations are happening at the same time
  - multiple tasks run & finish in overlapping time periods, in no specific order
- Parallelism: multiple tasks or subtasks of the same task run at the same time
- Any Combo of the 2:
  - Sequential execution - programs we've written so far
  - Concurrent but not parallel - multiple different programs running at the same time
  - Parallel execution - ???
  - Parallel and concurrent - ???

Reference:

- https://cs423-uiuc.github.io/fall22/slides/06-concurrency.pdf

Parallelism

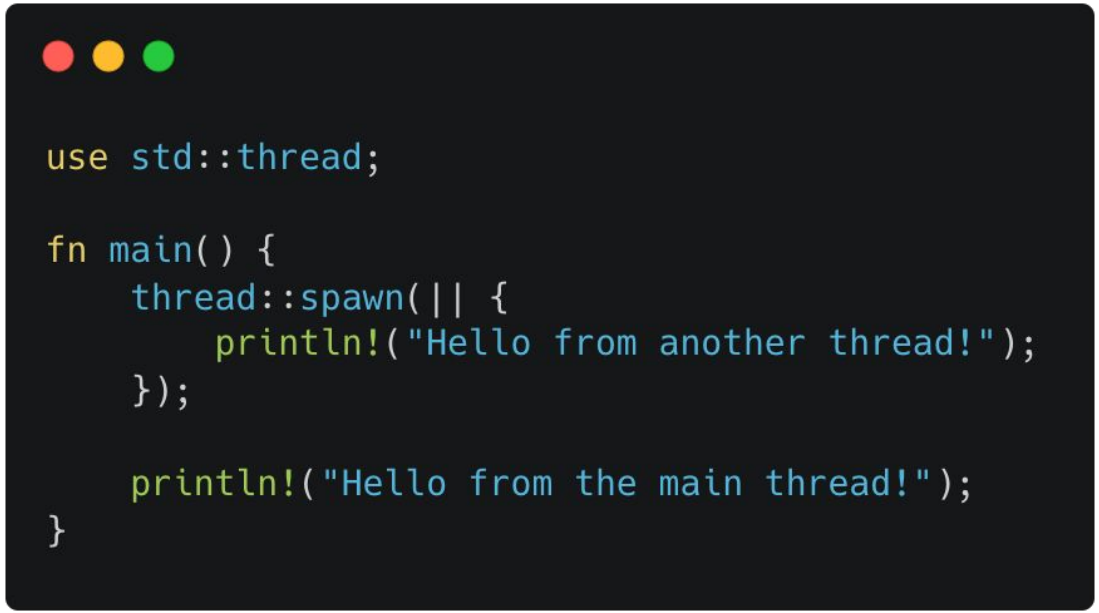
- Parallel Execution:
  - Split up your computation into different parts
  - Create threads to solve each part and solve at the same time
  - Combine the results at the end (if need be)
- Parallel and Concurrent:
  - Your computer!
  - Running multiple processes simultaneously
  - Each task might create its own threads to split up a problem

Reference:

- https://cs423-uiuc.github.io/fall22/slides/06-concurrency.pdf

Spawning a Single Thread

Spawning a Single Thread (BUGGY)



```rust
use std::thread;

fn main() {
    thread::spawn(|| {
        println!("Hello from another thread!");
    });

    println!("Hello from the main thread!");
}
```

Spawning a Single Thread

- Programs finish when the main thread finishes
  - If the main thread finishes while threads are still running, the program will still end
- Spawning a thread comes with a slight overhead
  - In a long running program, this cost is insignificant
  - In short programs like this, the cost is noticeable

Spawning a Single Thread (Better)

```rust
use std::thread;

fn main() {
    let handle = thread::spawn(|| {
        println!("Hello from another thread!");
    });

    // Wait for our thread to finish
    match handle.join() {
        Ok(_) => (), // do nothing if our thread finished successfully
        Err(e) => println!("Thread join error {:?}", e)
    }

    println!("Hello from the main thread!");
}
```

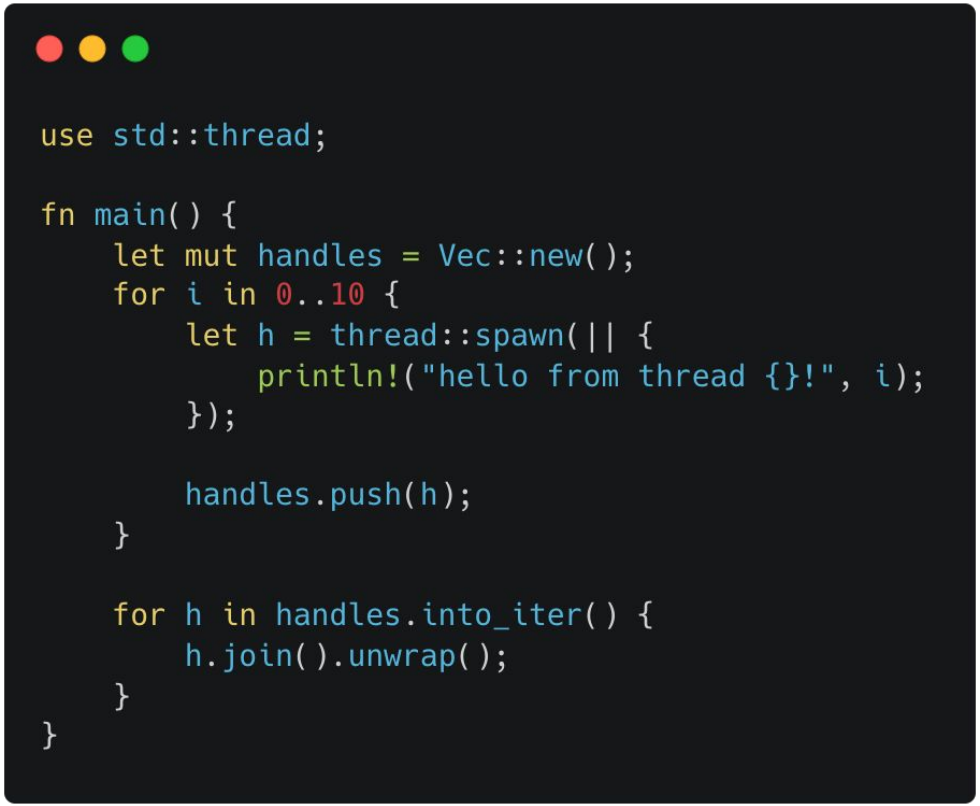
Joining Threads & Blocking Functions

- **JoinHandle**: A struct that gives you permission to join on a thread
  - Block the program until the thread terminates
- **thread::spawn()**: Spawns a single thread and returns a **JoinHandle** for it
- **join_handle.join()**:
  - Block the current thread until thread associated with the handle finishes
  - If the associated thread panics, **Err** is returned (otherwise **Ok** is returned)

Reference:

- https://doc.rust-lang.org/stable/std/thread/struct.JoinHandle.html
- https://doc.rust-lang.org/std/thread/fn.spawn.html

Spawning Multiple Threads (Doesn't Compile)



```rust
use std::thread;

fn main() {
    let mut handles = Vec::new();
    for i in 0..10 {
        let h = thread::spawn(|| {
            println!("hello from thread {}!", i);
        });

        handles.push(h);
    }

    for h in handles.into_iter() {
        h.join().unwrap();
    }
}
```

Joining Threads & Blocking Functions

- Use the **move** keyword to move data from the original thread to a new thread
- Threads own the data they create or acquire
- The **move** keyword means:
  - Either Transfer ownership of data to the new thread
  - Or copy (NOT Clone) any data that can be copied into the new thread
    - i.e. primitive types

Reference:

- https://doc.rust-lang.org/stable/std/thread/struct.JoinHandle.html
- https://doc.rust-lang.org/std/thread/fn.spawn.html

Spawning Multiple Threads (Better)

```rust
use std::thread;

fn main() {
    let mut handles = Vec::new();
    for i in 0..10 {
        let h = thread::spawn(move || {
            println!("hello from thread {}!", i);
        });

        handles.push(h);
    }

    for h in handles.into_iter() {
        h.join().unwrap();
    }
}
```

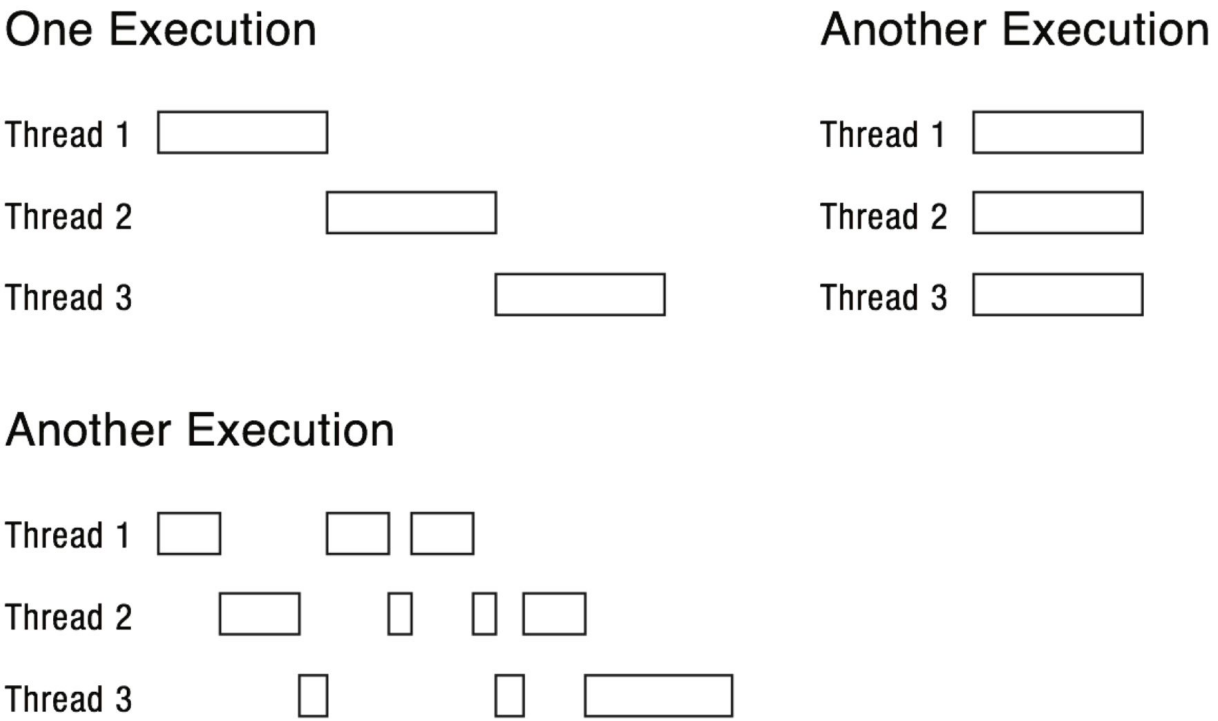
Hint:

- Use the **move** keyword to move data from the original thread to a new thread

Spawning Multiple Threads (Better)

Parallelism

Processor View



Reference:

- https://cs423-uiuc.github.io/fall22/slides/06-concurrency.pdf

That’s All Folks!