Class 9: Multi-Tasking Map

Action Items

PS2 is due Monday, September 30. The submission form for PS2 will be posted soon. As part of submitting PS2, you will schedule a short (10 minute) demo/review with one of the course staff. All team members are expected to participate in the demo, except in exceptional circumstances. At the demo, you should be prepared to show off your shell and will be asked to try some specific things in your shell and explain things about what happens, as well as answering questions about your design and implementation.

Sequential Map

```
struct Node {
    head : int,
    tail : Option<~Node>
}

type List = Option<~Node> ;

trait Map {
    fn mapr(&self, &fn(int) -> int) -> List;
}

impl Map for List {
    fn mapr(&self, f: &fn(int) -> int) -> List {
         match(*self) {
            None => None,
            Some(ref node) => { Some(~Node{ head: f(node.head), tail: node.tail.mapr(f) }) },
        }
    }
}
```

You should understand everything in this code. If you don't feel comfortable with functions that take other functions as inputs or return new functions as results and recursive data structions and procedures on them, you should read John McCarthy's *Recursive Functions of Symbolic Expressions and their Computation by Machine* (Part I) (*Communications of the ACM*, April 1960). If you prefer a more "colorful" presentation, I would recommend reading Chapter 4 and Chatper 5 from a highly-rated but lowly-ranked and rarely-used introductory computing textbook. You should be able to translate the program examples from the Scheme language used there into Rust.

Another interesting writing from John McCarthy is *Progress and its Sustainability*. Although, you might want to consider his predictive ability in light of this one also: *Networks Considered Harmful for Electronic Mail* (December 1989).

> *The reason why telefax will supplant email unless email is separated from special networks is that telefax works by using the existing telephone network directly. To become a telefax user, it is only necessary to buy a telefax machine for a price between $1,000 and $5,000 (depending on features) and to publicize one's fax number on stationery, on business cards and in telephone directories. Once this is done anyone in the world can communicate with you. No complicated network addresses and no politics to determine who is eligible to be on what network. Telefax is already much more widely used than email, and a Japanese industry estimate is that 5 percent of homes will have telefax by 1995 and 50 percent by 2010. This is with a $200 target price. . . . More generally, suppose the same need can be met either by buying a product or subscribing to a service. If the costs are at all close, the people who sell the product win out over those selling the service. Why this is so I leave to psychologists, and experts in marketing, but I suppose it has to do with the fact that selling services requires continual selling to keep the customers, and this keeps the prices high. I hope my pessimism about institutions is unwarranted, but I remember a quotation from John von Neumann to some effect like expecting institutions to behave rationally is like expecting heat to flow from a cold place to a hot place.*

Were institutions more rational than John McCarthy predicted, or is there some other reason why his prediction was so wrong?

Processes, Threads, and Tasks

What is difference between a process and thread?

Is a Rust task more like a process or a thread?

How can Rust tasks provide process-like memory isolation but without the expense of a traditional process?

Spawning Tasks

The `spawn` function takes in an owned function (which cannot capture any mutable, shared objects) and runs that function in a new thread:

```
    fn spawn(f: ~fn())
```

These two constructs are equivalent (the second is preferred for readability, but the first is better for understanding what you are doing):

```
spawn( | | {
    println("Get back to work!");
});

do spawn {
    println("Get back to work!")");
}
```

Channels provide a way for spawned tasked to communicate back to their parent:

```
   let (port, chan) : (Port<int>, Chan<int>) = stream();
   let val = node.head;
   do spawn {
      chan.send(f(val));
   }
   let newval = port.recv();
```

The `port.recv()` call with block until a sent value (from `chan.send(f.val)`) arrives.

Multi-Tasked Mapping

```
struct Node { head : int, tail : Option<~Node> }
type List = Option<~Node> ;

trait Map {
    fn mapr(&self, extern fn(int) -> int) -> List;
}

impl Map for List {
    fn mapr(&self, f: extern fn(int) -> int) -> List {
         match(*self) {
            None => None,
            Some(ref node) => {
                let (port, chan) : (Port<int>, Chan<int>) = stream();
                let val = node.head;
```

```
                do spawn {
                    let res = f(val);
                    chan.send(res);
                }
                let newtail = node.tail.mapr(f); // needs to move here!
                let newval = port.recv();
                Some(~Node{ head: newval, tail: newtail })
            }
        }
    }
}

fn collatz_steps(n: int) -> int {
    if n == 1 {
        0
    } else {
        1 + collatz_steps(if n % 2 == 0 { n / 2 } else { 3*n + 1 })
    }
}

fn find_collatz(k: int) -> int {
    // Returns the minimum value, n, with Collatz stopping time >= k.
    let mut n = 1;
    while collatz_steps(n) < k { n += 1; }
    n
}
```

Modern Processors

For a much more entertaining answer to the question of why the best laptop I could buy at Cavalier Computers yesterday only has four cores, read James Mickens' *The Slow Winter* (a hilarilous short essay, but with lots of big technical ideas in it)

> *Today, if a person uses a desktop or laptop, she is justifiably angry if she discovers that her machine is doing a non-trivial amount of work. If her hard disk is active for more than a second per hour, or if her CPU utilization goes above 4%, she either has a computer virus, or she made the disastrous decision to run a Java program. Either way, it's not your fault: you brought the fire down from Olympus, and the mortals do with it what they will. But now, all the easy giants were dead, and John was left to fight the ghosts that Schröinger had left behind. "John," you say as you pour some eggnog, "did I ever tell you how I implemented an out-of-order pipeline with David Hasselhoff and Hulk Hogan's moustache colorist?" You are suddenly aware that you left your poncho in the other room.*