Abusing Multi-Threading to Implement Generator Functions in C

Rushi Shah

March 4, 2017

Intro about what the project is

1 The typical way

setjmp/longjmp won't work because

Manually flipping stacks for execution (in assembly).

2 Why threads are appealing

Maintaining state among the different functions is a pain. Threads will manage that automatically, granted that you can get the right thread to run the right amount of code at the right time.

3 How I did it

3.1 Mutex

3.2 Conditional Variable

4 Why I didn't finish implementing it

4.1 Arbitrary amounts of generators

My original implementation was sufficient for creating one iterator and bouncing control back and forth between the two routines. However, extending it to an arbitrary amount of threads would have been extremely difficult with the setup I had.

4.2

Since the project was due in a week I needed an easier method of doing things. Turns out we had already been provided with a method for switching the stacks which was the way we

were expected to go about doing things. That way was , but it was a fun little experiment.

5 Conclusion

One of the reasons I chose to attend UT Austin (in the Turing Program) was because I knew that Edsger Dijkstra used to be a professor here so I knew it must be a serious research institution. I had always known Dijkstra as the guy who made Dijkstra's shortest path algorithm. But in researching for this project I learned that he was ALSO the dude who invented semaphores. Without him, concurrent programming would be useless because threads would not have an effective method for sharing resources and thus passing data back and forth between threads. AI and distributed computing are the two buzz-words nowadays, but they would be significantly further behind in their maturity if it wasn't for Dijkstra. Edsger Dijkstra is a god.