Modeling and Programming 2018-1: Sixth lab practice

Luis Daniel Aragon Bermudez 416041271

November 27th, 2017

Contents

Dijkstra, Synchronicity, Parallelism and Design Patterns.

Article and commentary

Round Robin

It's a scheduling algorithm for time sharing systems. It sequentially allocates the same processing quanta (small time quantities) to each process in the processing queue. (Arpaci-Dusseau, 2014)

Semaphore

A semaphore is a control measure and condition that is "used to control access to a common resource by multiple processes in a concurrent system such as a multiprogramming operating system" and it can either be an integer or an abstract data type (Wikipedia, 2017). "*The Little Book of Semaphores*" defines a semaphore as basically an integer with three differences:

1. When you create the semaphore, you can initialize its value to any integer, but after that the only operations you are allowed to perform are increment (increase by one) and decrement (decrease by one). You cannot read the current value of the semaphore.
2. When a thread decrements the semaphore, if the result is negative, the thread blocks itself and cannot continue until another thread increments the semaphore.
3. When a thread increments the semaphore, if there are other threads waiting, one of the waiting threads gets unblocked.

They were first thought of and described (although his definition has since been modified) by Dijkstra in 1962 or 1963 (Dijkstra, 1962 or 1963).

Shortest route using Dijkstra's algorithm

Dijkstra's algorithm is used to find the shortest paths between nodes in a graph. "Dijkstra's original variant found the shortest path between two nodes, but a different variant fixes a single node as the"source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree" (Wikipedia, 2017).

Next, we'll run the algorithm in order to find the shortest path between the node A and the node E on a weighted graph (Graph G), step by step:

First, we set the distance from A to the other nodes to infinity, and set it to zero for A:

Start > A:0_{A}, B:inf, C:inf, D:inf, E:inf

Then we check the ex-nodes of A and update their distance:

A > A:0_{A}, B:inf, C:5_{A}, D:inf, E:inf

A's now been visited (all its ex-nodes have been queued), we now move on to the *closest* queued node (in this case, node C) and update the distance to its ex-nodes if the resulting distance is shorter:

C > A:0_{A}, B:inf, C:5_{A}, D:7_{C}, E:inf

C's now been visited (all its ex-nodes have been queued), we now move on to the *closest* queued node (in this case, node D) and update the distance to its ex-nodes if the resulting distance is shorter:

D > A:0_{A}, B:13_{D}, C:5_{A}, D:7_{C}, E:11_{D}

D's now been visited (all its ex-nodes have been queued), E has been reached. We're finished. The shortest distance from A to E is 11.

E > A:0_{A}, B:13_{D}, C:5_{A}, D:7_{C}, E:11_{D}

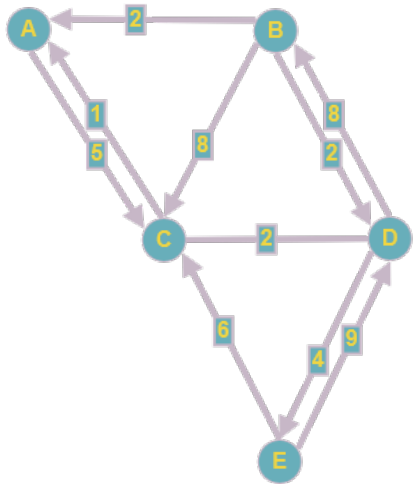

Figure 1: Graph G

Appalling Prose and the Shortest Path

I recently went out for lunch with a really smart friend, regardless of his long list of achievements, he's someone who's managed to use his knowledge in sociology and technology to find creative solutions to social issues. He's not only a fluent coder and sysadmin he's also an active member of the UNAM's School of Political and Social Sciences.

Among other things we talked about Computer Science, particularly we discussed whether CS should be considered scientific. We had both read Bruce Sterling's interview with Peter Denning, where Denning reflected on this particular subject:

> "The most difficult objection has been that computing can't be a science because science deals with natural phenomena, whereas computers are manmade. They say that, at best, computing is a science of

> the artificial, not a real science. (...) In 2004 I sat down and carefully checked how computing does or does not satisfy all the accepted criteria of being a science. These criteria include an organized body of knowledge, a track record of non-obvious discoveries, an experimental method to test hypotheses, and an openness to any hypothesis being falsified. (...) I saw that we could check off every one of the accepted criteria."

Nevertheless, we had really different points of view regarding the article. I imagine Dijkstra may have had a similar discussion at some point during his pioneering quest to transforming the scientific community's view on discrete and finite problems. What Dijkstra did was extremely innovative and his insistence, as well as his genius, were crucial to CS's development as a discipline.

Proving that something can (or can't) be done used to be the goal line but we soon found out that, when you *really* need to solve a problem, it might just be half the way there. Brilliant people like Dijkstra and Turin had to come up with algorithmic solutions to give life to the era of computers. The process of designing electronics and systems was just as important as the electronics themselves.

Semaphores, the dining philosophers' problem and Dijkstra's multiple contributions to parallelism, system design, and CS in general will never be forgotten, and to this day they keep influencing not only the field but the whole world.

State Design Pattern implementation

The state design pattern is a behavioral pattern; it can be described as an object-oriented state machine that "allows an object to change its behavior when its internal state changes" (SourceMaking, 2017). Its implementation consists of creating a State interface that includes all the state-dependent methods of the wrapper class, the wrapper class has a current state (context) and a way to change between states, which implement the state interface. The state transitions are implemented by each state.

Included, there's a small implementation of a Dog that uses this design pattern and has the following state graph:

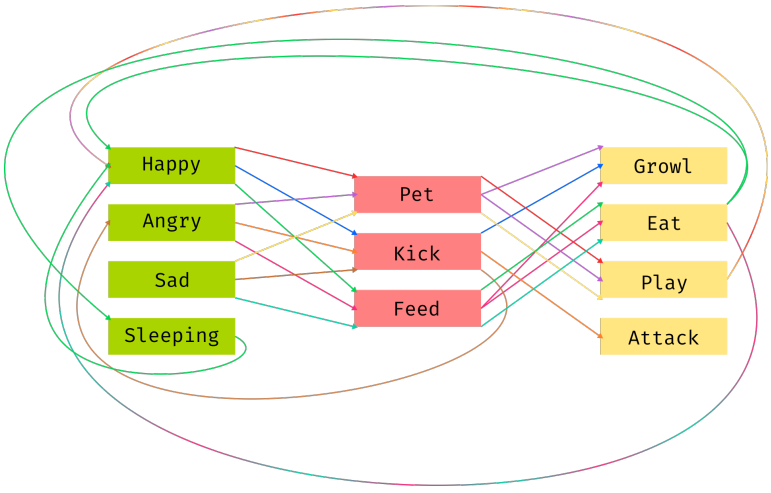

Figure 2: State diagram of the Dog class. The green boxes represent the valid states, the red boxes represent the possible interactions and the yellow boxes represent the dog's possible reactions.

The following rules are also coded in:

- If the dos is angry and you try to pet it or feed it, it will growl at you. You'll have to pet him or feed him 3 times before he becomes happy and actually eats the food or plays with you.
- If the dog is angry and you kick it, it will bite you and the program will close (via System.exit(1)).
- If the dog is happy and you feed it, it will have a 50% of becoming asleep. If the dog sleeps, it will continue to do so for 5 seconds, during which you will not be able to interact with it, after waking up the dog will become happy.

- If the dog is sad and you kick it three times, it will become angry.

The program consists of two packages:

fciencias.clutil

This package contains the utility class CLUtil, which provides useful terminal decorators and input validation methods.

fciencias.myp.main

This package includes the demo via the implementation of a dog and a simple menu that allows the user to interact with said dog.

Building and running the program

The program can be built using ant, the available commands are described bellow. If you're on Linux and have python installed, then running the following command from the project's main directory will be enough to build and run the program: `ant executable; ./dog`. After having run `ant executable` once, you'll only need to run `./dog` to launch the program.

The included `build.xml` provides the following commands:

1. `ant build`, compiles the program to `out/jar/Dog.jar`.
2. `ant doc`, generates the program's documentation and puts it inside `doc/`.
3. `ant run`, compiles the program, creates the jar and runs the application (using the awful ant logger). This is not the recommended way to run the program, use the following command to create an executable instead:
4. `ant executable`, creates a runnable file `./dog` that uses a python script to run the actual jar file.
5. `ant clean`, deletes all files and folders except for `src/`, `LICENSE`, `build.xml`, `README.md`, `.gitingore` and `README.pdf`.

Bibliographgy

- "Operating Systems: Three Easy Pieces (Chapter 7: Scheduling Introduction)" - *Arpaci-Dusseau*, 2014
- "Semaphore (programming)" - *Wikipedia*, 2017
- "The Little Book of Semaphores" - *Downey, Allen B.*, 2016
- "Over de sequentialiteit van procesbeschrijvingen" - *Dijkstra, Edsger W.* 1962 or 1963
- "Dijkstra's algorithm" - *Wikipedia*, 2017
- "State Design Pattern" - *SourceMaking*, 2017

Acknowledgements

For more information on the tools used to build, create and run this program refer to the following links:

- Apache Ant was used to create the build script.
- Python was used for the execution script.
- JetBrains' IntelliJ IDEA was used as the primary editor.
- Graph Online was used to create the directed graph G.