Distributed Observation Worm (DOW)

Mandatory assignment 2 for INF-3203, Spring 2017.

Due date: Thursday 4 May, 2017

Introduction

Computer worms were described in 1982 [1]. Worms were regarded as "distributed computations", and several issues with such computations were analysed, including how they can sometimes go wrong and create problems for many computers.

This project will develop a worm abstraction and a concrete functioning prototype worm system and a basic worm.

A worm comprises one or several worm *segments*. A segment is a program. The segments are typically identical. The segments are located on multiple computers, typically one segment per worm per computer.

A segment can be executed as one or more processes or threads.

A worm typically has a *payload*. The payload can be a mission like collecting data from the environments where the segments are executing, increasing robustness for an application, harvesting of CPU cycles, and be used by public and private purposes.

Architecture

A worm system **architecture** comprises a set of **functionalities & characteristics** that taken together defines the worm **abstraction**:

- **Worm State monitoring**: A worm monitors its own state to determine how many of its segments are alive.
- **Spreading**: A worm initializes on a single computer. Spreading to other computers are done to distribute segments and to stay alive.
- **Stay alive**: Worm segments can terminate for various reasons (crashes, explicit kill signals) or become partitioned (for example because of network partitioning). The worm will try to stay alive by spreading new segments when segments disappears. A worm has a target number of segments that it will try to keep alive.
- **Kill**:
    - o **Two types of “kill” are needed**

        - **Intentional**: A worm should have a way of committing suicide. A suicide could be the result of a decision internal to the worm, or because of an explicit kill command from the Command & Report Central. Suicide and killing are in particular useful if the worm starts to spread uncontrollably, or behaves in ways not anticipated.
        - **From outside**: To experiment with a worm, a kill mechanism on individual segments are useful. This allows the Command & Report Central to selectively kill off segments on any computer. As a result, the Stay Alive mechanism should kick in and try to keep the worm as a whole alive.
- **Intra-worm inter-segment interaction**: The worm segments should have a way of interacting with each other.
- **Inter-worm interaction**: Separate worms should have a way of interacting with each other (and with a *command & report central*.)

Design

The **design** details how the worm abstraction can be structured into two “sides”, worm side and worm gate side.

Each mechanism and functionality can be a client, a service, a server, a process, a thread (*think about what to use*).

- **Worm**
    - **Detect the state of the worm**
        - A segment collects data to determine how many of its sibling segments are running.
        - A segment keeps track of where its sibling segments are located.
    - **Stay alive**
        - If a segment detects a reduction in the number of segments of the worm, it can invade a computer with a new segment.
        - A new segment will let Command & Report Central (see below) poll it to find out about its existence and where it is at.
    - **Kill**: When a kill command is issued by the control centre, the worm should try to commit suicide no matter what.
- **Payload**
    - In the original worm paper, the worm itself is used as a platform for some other *payload* application that would do something useful. They experimented with several payload including a distributed alarm clock program and distributed animation rendering.
    - For the assignment, the worm itself is the payload. It will have to stay alive while an outside force is actively trying to kill it. The Command & Report

Central will kill individual segments with some frequency. The worm's task is to try to estimate how frequently its segments are being killed. (*NB: this is not the worm suicide command-- this will be the host environment killing the worm processes from the outside*). The frequency estimate must be made available to the CRC (the CRC will poll the segments of a worm for it).

- **Worm gate**
  - A *worm gate* is a service/server/daemon/process that executes on each computer where segments can reside.
  - For a malicious worm this could be some kind of security exploit. For a benign worm it could be a service that intentionally lets the worm in and runs it. For this assignment it is a service that we provide.
  - The worm gate provides a set of services to segments and the Command & Control Central (*for this project we will keep the quantity and functionality of services very limited.)*.
    - **Monitoring service**: The monitoring service keeps track of the state of all segments on the gate computer. We will limit the state to be which segments are running on the gate computer.
    - **Spreading service**: The worm gate allows a ***remote*** worm segment to request upload and start-up of a segment.
    - **Kill service**: A worm gate knows of all segments executing on its computer and can kill them. The Command & Report Central will use this to selectively kill segments.
    - **Topology service**: The worm gate has a map of the topology of computers reachable from the computer where the worm gate resides. When a segment wants to communicate with its sibling segments, it must first check the topology map to determine if they are reachable.

Worm Command & Report Central (CRC) side

A *Command, Control & Report* Central is where a user or a program can interact with a worm. In general, this can be through a shell or an application custom designed for the task.

We will provide you with a simple CRC and the interface your worm segments must adhere to.

The ***Command*** part of the CRC receives commands from a user, and sends them to a worm. All segments should be able to receive commands and act upon them.

The commands a worm must expect are:

- **Kill worm**: Command to kill a **worm**.
    - The CRC will issue kill commands to a worm with some frequency. We will increase the frequency during the demo to exercise the Stay Alive characteristics of the worm.
- **Change** number of segments in the worm (±number)

The CRC can also request a worm gate to **kill** a **segment** running on the gates computer. This is not a command seen by a worm. However, the worm should discover that it is losing segments.

The ***Report*** part of the CRC is where data **about** and **from** a worm segment is presented.

How to interact with the CRC

See the README in the code repository for details about the APIs of the different components.

Limitations

- The worm should consist of **fully autonomous segments** and contain **no centralized control** whatsoever.

- Do not use the cluster's shared filesystem for communication. Your worm segments must communicate over the network, and they must use the worm gate to spread.

- You should use rudimentary communication mechanisms. If some communication library seems to solve a lot of problems for you, you're probably not allowed to use it. Do ask us before using such communication mechanisms and libraries.

Practical hints

- The worm gate and example code are written in Go (aka golang). You may use any language you like, but we encourage you to give Go a try.

- We recommend doing the worm segments as independent processes interacting using the local network.

Environment

You will use the teaching cluster (uvrocks) for testing, debugging and demonstrating your worm.

See the README in the code repository for details about how to run and kill the components.

Groups

We encourage you to work in groups of two. If you want to work in larger groups (or individually) you can ask the TAs or the professor, but bear in mind that larger groups are expected to do more work.

Workflow

You should zip (or tarball) your code and report and upload them to Fronter. However, we will publish the worm gate and example code as Git repository and we encourage you to use Git for your work. If we need to update the example code for any reason, we will push out the changes to the Git repo and you can fetch and merge them into your code.

Deliverables

What you have to do:

- A prototype of the parts of the worm abstraction that we don't provide you with.
- A prototype worm as described above.
- The worm must interact with the Command & Report Central as described above.

You must provide us with:

- Your source code and instructions on how to run it (update the README).
- A three-to-four-page report (PDF format) describing your solution, its design and implementation, and any unexpected issues/lessons learned while working on this project.
  - Bonus points if you've designed experiments doing performance measurements on your worm (think about what you want to measure).
  - Also, we would like to hear your ideas for possible applications of a worm. Could this concept have any beneficial uses in the real world? Or is it just an interesting toy at best or malicious exploit at worst?

Presentations

You are required to give a brief demo of your worm. Like the assignments with INF-3200 last semester, we will have some visualization/test code running (the worm gates + command and control center) and we will ask you to run your code and to give us a brief explanation of your design and the challenges you faced.

Important Dates

- Hand-out: Monday 2017-03-13
- Hand-in and presentations: Thursday 2017-05-04
- May the Fourth be with you.

Advice

- Start early—fail early.
- Talk to one another, your fellow students can be valuable resources.
- Use different ports so you don't clash with the other students.
- If your worm is out of control and you need an emergency brake, you can kill off the worm gates to keep it from spreading. However, note that this will not be accepted as a valid solution for the worm suicide requirement.

Literature/References

1. “The "Worm" Programs – Early Experience with a Distributed Computation”, John F. Shoch and Jon A. Hupp, in Communications of the ACM, vol. 25 no. 3, 1982, pages 172– 180.
2. Another history lesson: The Morris Worm caused some serious trouble in 1988 because of an overzealous spreading mechanism.
3. The novel that inspired Shoch and Hupp: The Shockwave Rider by John Brunner