High Performance Programming and Systems

Assignment Five

Network Messaging

Set: *21st of December 2023*
Due: *7th of January 2024 @ 23:59 CEST*

Synopsis:
Send network messages in the correct order to avoid deadlock.

1 Introduction

This assignment is meant to test three competences:

1. Writing networked programs
2. Managing asynchronous communications
3. Demonstrating system correctness

Keep this in mind when working on the code, and particularly when choosing what to focus on for your report. Make sure to read the entire text before starting. In particular, read section 5 to understand the code handout.

2 The Santa Problem

You will be implementing several solutions to one of the many 'Santa Problems'. This should test your ability to program networked applications, as well as to reason about the behaviour of concurrent processing. The Santa Problem in question was originally set out by John Trono (`https://santaclausproblem.cs.unlv.edu/`), and is intended as learning exercise in multiprocess communication. We are going to adapt it slightly, as it sets a more complex problem than we really have time to solve. Our adapted problem is:

Santa Claus sleeps in his shop up at the North Pole, and can only be wakened by either all nine reindeer being back from their year long vacation *on the beaches of some tropical island in the South Pacific, or by some elves who are having some difficulties making the toys. One elf's problem is never serious enough to wake up Santa (otherwise, he may never get any sleep), so, the elves visit Santa in a group of three. It is assumed that the reindeer don't want to leave the tropics, and therefore they stay there until the last possible moment. The penalty for the last reindeer to arrive is that it must get Santa while the others wait in a warming hut before being harnessed to the sleigh.*

This gives us the description of the 3 process types in our system

1. Santa mostly just waits for other processes to tell it to either deliver presents with all the reindeer, or solve problems with a subset of elves. Once he has done either he will go back to waiting for the next notification.

2. Reindeer will sleep for a random amount of time to simulate going on holiday. Once all have woke it is assumed it is Christmas Eve-eve and it is time for all reindeer and Santa to deliver presents. The reindeer will then go holiday again, and so on, in an infinite loop.

3. Elves will sleep for a random amount of time to simulate building toys. Once three have woken, the problems are serious enough to go and wake Santa. All three elves and Santa will then solve the problems, and the elves can back to work in an infinite loop.

You will be writing a series of small programs that simulate these interactions in a more complete manner. Most of the code has been provided, so you can just focus on how the processes interact. Note that the code has been provided so that you can either run each process individually (either on one machine or many), or so that it can be run as one 'block'.

3 Programming Tasks

There are three programming tasks to complete, each of which will be solving the Santa problem in a slightly different manner.

3.1 The Naive Solution

Here your task is to complete the naive solution. In this, each of the reindeer and elf processes will message Santa directly when they have returned or developed problems. Santa will count them in and once each have assembled will either deliver presents or solve problems. The program can be started by calling the 'naive_santa_problem.py' like so:

```
$ python3 naive_santa_problem.py 9 12 3
```

This will start a program with 9 reindeer, 12 elves and where the elves solve problems in groups of 3. You can also start each of the components separately by calling each of the following lines in a different terminal:

```
$ python3 naive_elf.py 100 127.0.0.1 1111
      127.0.0.1 9999
$ python3 naive_reindeer.py 200 127.0.0.1 2222
      127.0.0.1 9999
$ python3 naive_santa.py 127.0.0.1 9999 9 3
```

These lines will each start an elf, reindeer and Santa. In these cases the elf has an ID number of 100 and is hosted at 127.0.0.1:1111, the reindeer has an ID of 200 and is hosted at 127.0.0.1:2222, while Santa is at 127.0.0.1:9999 and expects 9 reindeer and groups of 3 elves. You will need to manually set unique port numbers for each elf and reindeer instance if you wish to start multiple of them on your local machine.

Most of the files have been handed out in a complete state. The only part that needs finished is the naive_santa.py. You will find on line 59 a TODO, where you must implement the handling of messages from the elf processes. A strong hint is given from the code for the reindeer, as the elves can be solved in a very similar manner, though with the key difference that Santa must respond to groups of 3 elves (no more, no less) rather than all the reindeer.

3.2 The Socketserver Solution

The naive solution used manual socket programming, which is not a common design pattern in modern Python programming. A more robust solution is to use a socketserver class, that can automatically enable concurrent processing of many simultaneous connections.

Outwardly this problem has the same structure as the naive program, with reindeer and elves messaging santa. It can therefore be started in a very similar manner:

```
$ python3 socketserver_santa_problem.py 9 12 3
```

This will start a program with 9 reindeer, 12 elves and where the elves solve problems in groups of 3. You can also start each of the components separately by calling each of the following lines in a different terminal:

```
$ python3 socketserver_elf.py 100 127.0.0.1 1111
      127.0.0.1 9999
$ python3 socketserver_reindeer.py 200 127.0.0.1 2222
      127.0.0.1 9999
$ python3 socketserver_santa.py 127.0.0.1 9999 9 3
```

These lines will each start an elf, reindeer and Santa. In these cases the elf has an ID number of 100 and is hosted at 127.0.0.1:1111, the reindeer has an ID of 200 and is hosted at 127.0.0.1:2222, while Santa is at 127.0.0.1:9999 and expects 9 reindeer and groups of 3 elves. You will need to manually set unique port numbers for each elf and reindeer instance if you wish to start multiple of them on your local machine.

Similarly to the naive program, most of the handed out code is complete. You just need to complete socketserver_santa.py. A TODO has been included on line 15 indicating where you may wish to begin. Your task is to implement the socketserver handler function, that will be called concurrently for each incoming message. You should find that most of this code will be the same as the handling in the naive_santa.py, though you may have to make small modifications to account for any concurrency issues that arise.

3.3 The True Solution

Those of you who read the original Santa problem closely will note that Santa is not meant to be doing anything until all the reindeer or a subset of elves are ready. Our previous solutions didn't do this, as Santa was waking up to receive messages and then decide if he should sleep again or not. A more correct solution would have the reindeer and elves assembling somewhere else. Therefore we shall introduce two new processes, the stable and the porch.

The idea is that when the reindeer return from their holidays, they will not inform Santa directly, but will inform the stable process. The stable will then count all reindeer as they return. A second complication is that it is the job of the last reindeer to inform Santa that they are ready to go. Therefore, once the stable has determined that all reindeer are present, the stable must inform the last reindeer to arrive that it is last. This last reindeer will then inform all other reindeer and Santa that they can deliver presents. Do note that the reindeer in this program do not know the address of either Santa, or the other reindeer. Therefore as part of the stables notification message to the last reindeer, it must include a list of all reindeer and Santa addresses. You make look at how addresses were sent in the earlier programs for how this might be one.

The elves function very similarly, though they have a separate porch process to count them in. Once enough of them assemble then the first to arrive is notified that it must inform the other elves and Santa that it is time to sort problems. The complete program can be started like so:

```
$ python3 true_santa_problem.py 9 12 3
```

This will start a program with 9 reindeer, 12 elves and where the elves solve problems in groups of 3. You can also start each of the components separately by calling each of the following lines in a different terminal:

```
$ python3 true_elf.py 100 127.0.0.1 1111
      127.0.0.1 9999
$ python3 true_reindeer.py 200 127.0.0.1 2222
      127.0.0.1 9999
$ python3 true_santa.py 127.0.0.1 9999 9 3
```

These lines will each start an elf, reindeer and Santa. In these cases the elf has an ID number of 100 and is hosted at 127.0.0.1:1111, the reindeer has an ID of 200 and is hosted at 127.0.0.1:2222, while Santa is at 127.0.0.1:9999 and expects 9 reindeer and groups of 3 elves. You will need to manually set unique port numbers for each elf and reindeer instance if you wish to start multiple of them on your local machine.

Your task is to complete the network message interactions between the stable and reindeer, and the porch and elves. You will therefore need to modify true_elf.py, true_reindeer.py, stable.py and porch.py. Each has a TODO statement as a starting point, though you may be able to copy various code sections from previous programs. You must still account for any concurrency issues that arise.

3.4 The Checkin Program

An additional program has been provided, that may be of assistance. You do not have to use it, and using it does not count as testing or proof of correctness.

```
$ python3 shared.py
```

By calling the above, a simple server will be started that any process can send a message to, containing a unique ID. Each time the server receives a message it will print all the IDs it has received a message from, and how long ago it last heard from them. This can be used to identify possible deadlocks if you run a system with many reindeer and elves, especially if you use very small holiday and work times. If any of the programs are started via that X_santa_problem.py scripts, they will automatically check in with the checkin program (if it is running), though you may need to make more setup if starting processes individually.

4 The report (25%)

For this task, the report should cover at least the following:

- Describe how you completed the naive problem. This will probably be a very brief description.
- Describe how you completed the socketserver problem. Is your solution free from race conditions and deadlocks? You must indicate what testing/analysis you have done, and may wish to include any diagrams or psuedo-code you find necessary.
- Describe how you completed the true problem. Is your solution free from race conditions and deadlocks? You must indicate what testing/analysis you have done, and may wish to include any diagrams or psuedo-code you find necessary.

5 Handout/Submission

Alongside this PDF file there is a tarball containing a framework for the assignment.

```
$ tar xvf a4-handout.tar.gz
```

You will now find a `a4-handout` directory containing the following:

`naive_elf.py`

An implementation for the elf process, within the naive problem. This should not need to be modified.

`naive_reindeer.py`

An implementation for the reindeer process, within the naive problem. This should not need to be modified.

`naive_santa_problem.py`

A script to set up the naive problem, by starting a number of reindeer and elves. You will need to modify this to solve the naive problem. Details of how to do so can be found in Section 3.1.

`naive_santa.py`

An implementation for the Santa process, within the naive problem. This should not need to be modified.

`porch.py`

An implementation for the porch process, within the true problem. You will need to modify this to solve the true problem. Details of how to do so can be found in Section 3.3.

`shared.py`

A variety of shared configuration options and functions, used throughout. You will need to modify this to solve the true problem. You may also use this to start a checkin process, to help identify deadlocks. Details of how to do so can be found in Section 3.4.

`sockerserver_elf.py`

An implementation for the elf process, within the sockerserver problem. This should not need to be modified.

`sockerserver_reindeer.py`

An implementation for the reindeer process, within the sockerserver problem. This should not need to be modified.

`socketserver_santa_problem.py`

A script to set up the socketserver problem, by starting a number of reindeer and elves. You will need to modify this to solve the socketserver problem. Details of how to do so can be found in Section 3.2.

`socketserver_santa.py`

An implementation for the Santa process, within the socketserver problem. This should not need to be modified.

`stable.py`

An implementation for the stable process, within the true problem. You will need to modify this to solve the true problem. Details of how to do so can be found in Section 3.3.

`true_elf.py`

An implementation for the elf process, within the true problem. You will need to modify this to solve the true problem. Details of how to do so can be found in Section 3.3.

`true_reindeer.py`

An implementation for the reindeer process, within the true problem. You will need to modify this to solve the true problem. Details of how to do so can be found in Section 3.3.

`true_santa_problem.py`

A script to set up the true problem, by starting a number of reindeer and elves. You will need to modify this to solve the true problem. Details of how to do so can be found in Section 3.3.

`true_santa.py`

An implementation for the Santa process, within the true problem. This should not need to be modified.

6 Deliverables for This Assignment

You should submit the following items:

- A single PDF file, A4 size, no more than 5 pages, describing each item from report section above
- A single zip/tar.gz file with all code relevant to the implementation

7 Handing In Your Assignment

You will be handing this assignment in using Absalon. Try not to hand in your files at the very last-minute, in case the rest of the students stage a DDoS attack on Absalon at the exact moment you are trying to submit. **Do not email us your assignments unless we expressly ask you to do so.**.

8 Assessment

You will get written qualitative feedback, and points from zero to 4. There are no resubmissions, so please hand in what you managed to develop, even if you have not solved the assignment completely.