|  |  |
| --- | --- |
|  | **Royal Institute of Technology**  **MSc. Software Engineering of Distributed Systems** |

ID2203 Distributed Systems Advanced Course

Homework 1

|  |
| --- |
|  |
|  |
| Andrei Shumanski | Trigonakis Vasileios |
| andreish@kth.se | vtri@kth.se |
| 0046707761992 | 0046707694420 |

*Stockholm 2010*

Table of Contents

[Exercise 1 4](#_Toc253661265)

[Experiments 4](#_Toc253661266)

[Observations 4](#_Toc253661267)

[Conclusions 6](#_Toc253661268)

[Exercise 2 7](#_Toc253661269)

[Experiments 7](#_Toc253661270)

[Observations 8](#_Toc253661271)

[Conclusions 9](#_Toc253661272)

[Exercise 3 10](#_Toc253661273)

[Observations 11](#_Toc253661274)

[Conclusion 11](#_Toc253661275)

[Exercise 4 12](#_Toc253661276)

[Experiments 12](#_Toc253661277)

[Observations 13](#_Toc253661278)

[Conclusions 13](#_Toc253661279)

[Question 1 14](#_Toc253661280)

[Question 2 15](#_Toc253661281)

[Question 3 16](#_Toc253661282)

# Exercise 1

## Experiments

We tried the PFD with both 2 and 4 nodes in fully connected topologies.

Topology topology1 = **new** Topology() {

{

node(1, "127.0.0.1", 22031);

node(2, "127.0.0.1", 22032);

link(1, 2, 4000, 0).bidirectional();

}

};

Topology topology2 = new Topology() {

{

node(1, "127.0.0.1", 22031);

node(2, "127.0.0.1", 22032);

node(3, "127.0.0.1", 22033);

node(4, "127.0.0.1", 22034);

defaultLinks(1000, 0);

}

};

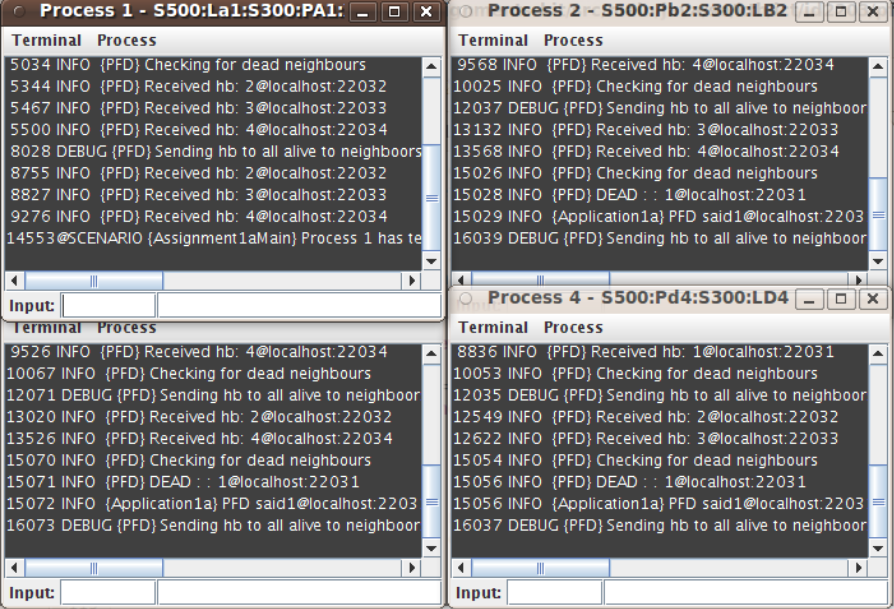
With the following values of γ and δ:

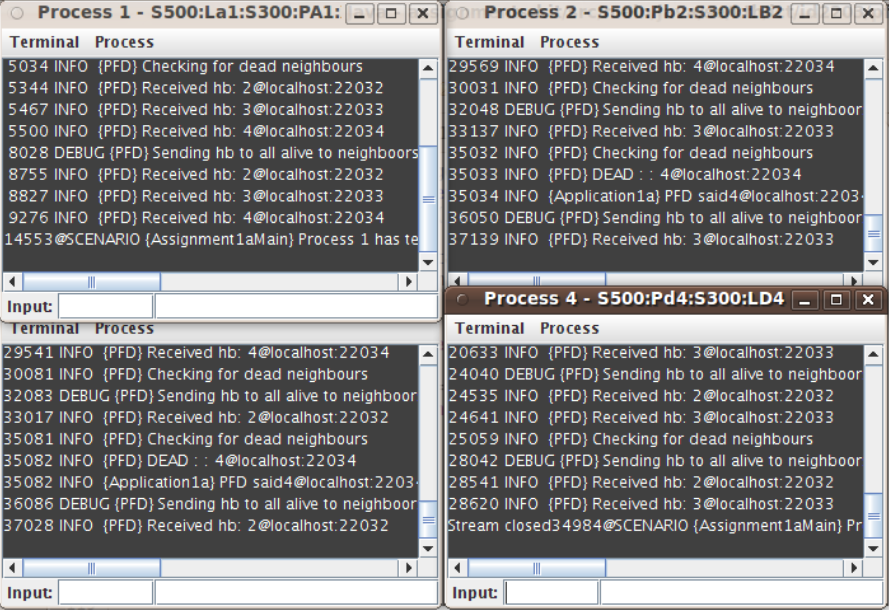
**private** **static** **long** *GAMMA* = 4000;

**private** **static** **long** *DELTA* = 1000;

## Observations

The PFD worked as intended, reporting the crash of a node as you can observe in the following screenshots:





## Conclusions

Our test showed that the algorithm and the implementation of the PFD works correctly and is consistent to the completeness property of it.

# Exercise 2

## Experiments

We tried the EPFD with both link delay shorter than the initial heartbeat period and longer than this. Also, in the first case, we killed one process to test that the other PFD is detecting the failure.

Topology topology1 = **new** Topology() {

{

node(1, "127.0.0.1", 22031);

node(2, "127.0.0.1", 22032);

link(1, 2, 3213, 0).bidirectional();

}

};

Topology topology2 = **new** Topology() {

{

node(1, "127.0.0.1", 22031);

node(2, "127.0.0.1", 22032);

link(1, 2, 4000, 0).bidirectional();

}

};

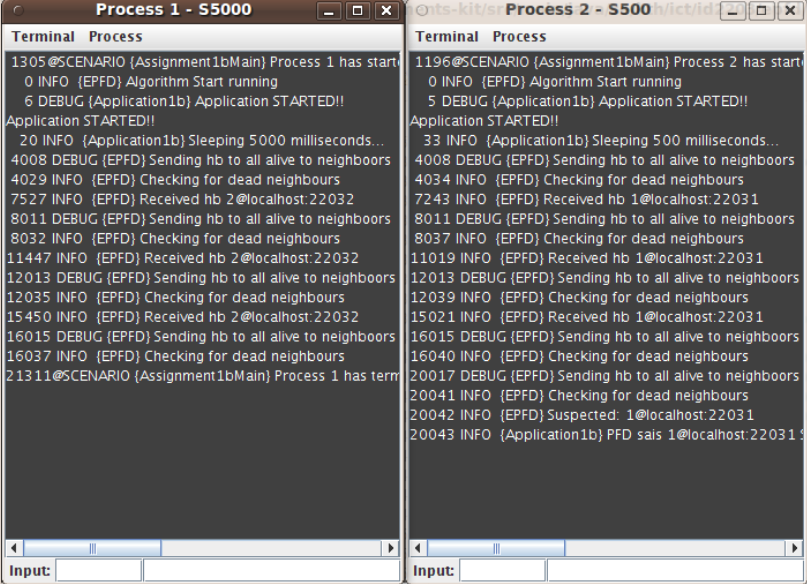
With the following values of TimeDelay and Δ:

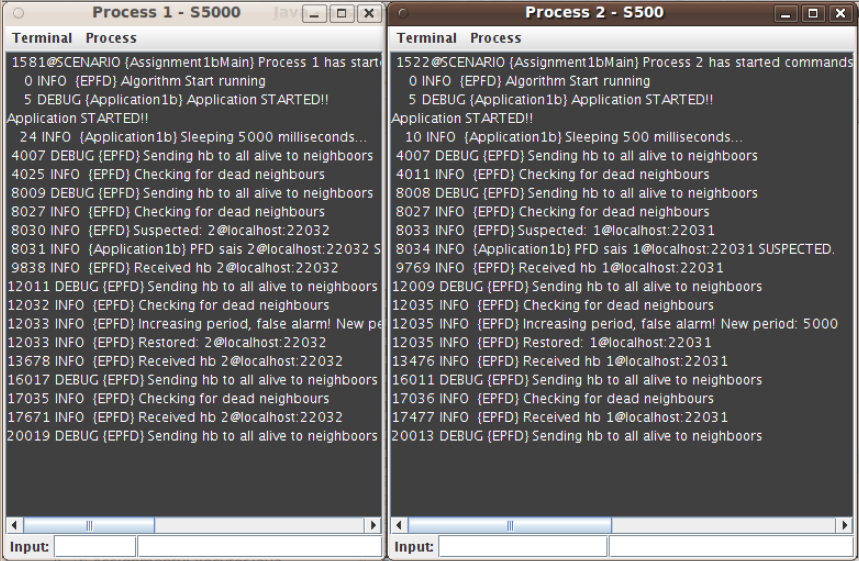
**private** **static** **long** *TIMEDELAY* = 4000;

**private** **static** **long** *DELTA* = 1000;

## Observations

The EPFD worked as intended, reporting the crash of a node in the 1st case and stabilizing in the 2nd case, as you can observe in the following screenshots:





## Conclusions

Our test showed that the algorithm and the implementation of the EPFD in the specific test cases are not that good. What is happening is that after the 1st stabilization of the period time, the heartbeat events keep coming every TimeDelay ms, so a second stabilization does not happen. Actually, this is not exactly a problem, because the EPFD correctly do not suspect the nodes, since they are not crashed.

# Exercise 3

In this exercise we used EPFD with FairLoss links to simulate loss of the messages. We used the following topology:

Topology topology1 = **new** Topology() {

{

node(1, "127.0.0.1", 22033);

node(2, "127.0.0.1", 22034);

link(1, 2, 1500, 0.7).bidirectional();

}

};

Latency = 1500 Ms;

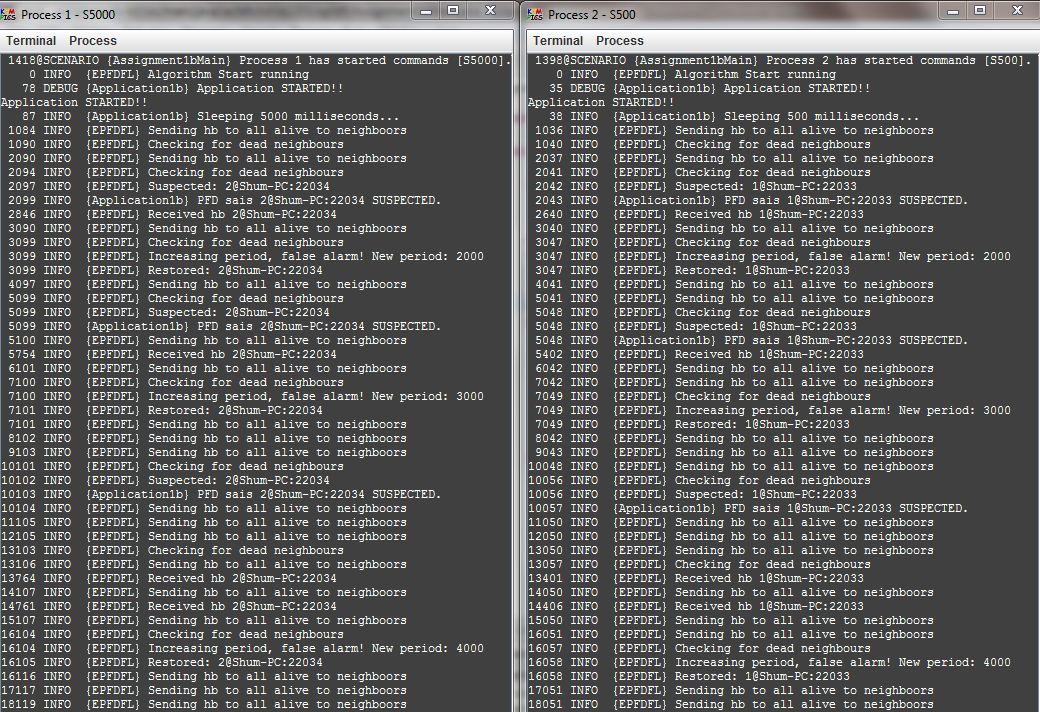
LossRate = 0.7;

TimeDelay = 1000 Ms;

Delta = 1000 Ms.

We used big loss rate to make adjustment of TimeDelay more often and visible.

## Observations



## Conclusion

We see that in this scenario TimeDelay increases to accommodate large delays when messages are lost. We observed that the larger is probability of loosing messages the faster grows TimeDelay. In the run shown above after 137 second TimeDelay increased to 8000 Ms.

# Exercise 4

## Experiments

We implemented the doRecover method, that is being run when a process recovers from a failure. It just prints an informational message to the logger. The topology and scenario that we used:

Topology topology1 = **new** Topology() {

{

node(1, "127.0.0.1", 22031);

node(2, "127.0.0.1", 22032);

link(1, 2, 3213, 0).bidirectional();

}

};

Scenario scenario3 = **new** Scenario(Assignment1bMain.**class**) {

{

command(1, "S500:Lmsg1:S6000:X").recover("R:S500:Pmsg3:S500:X", 10000);

command(2, "S500");

}

};

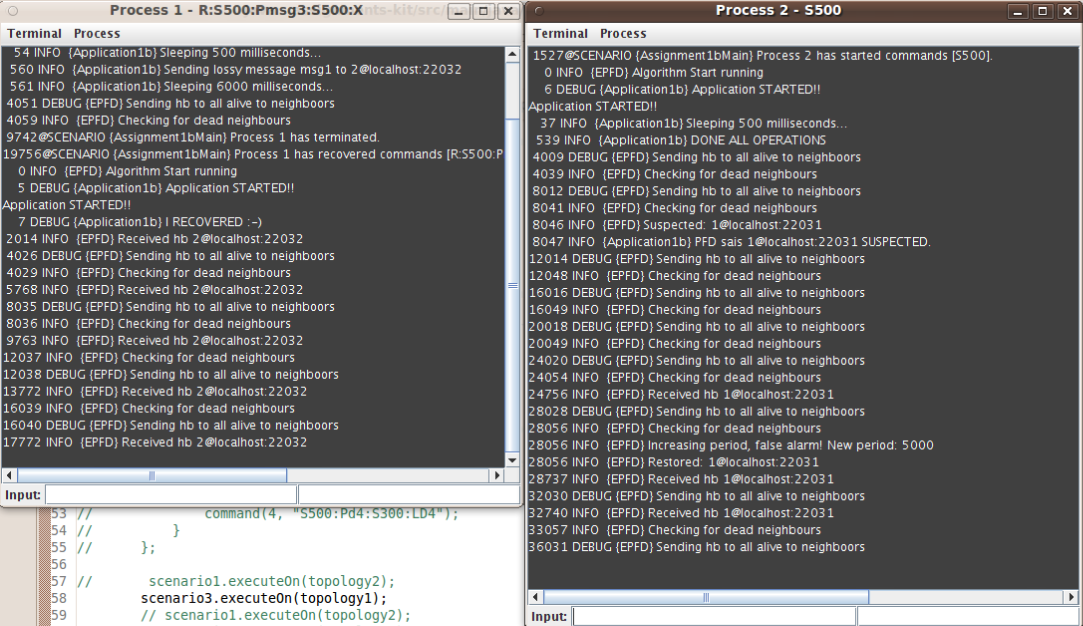
With the following values of TimeDelay and Δ:

**private** **static** **long** *TIMEDELAY* = 4000;

**private** **static** **long** *DELTA* = 1000;

## Observations

The results can be seen in the following screenshot:

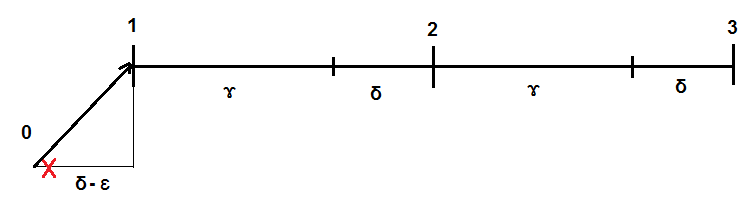


## Conclusions

As we expected, the EPFD correctly suspected the process that failed. This is absolutely normal, since in exercise 2 we tested that the EPFD detected a crashed node. Moreover, we can see that after the node recovers, it starts sending heartbeats again, and the other node calls a “false alarm” about the suspected event. Finally, we can conclude that the nodes that suspect the crashed node, when this node recovers, they are not able to distinguish between a node failure and recovery, an omission failure or just if the period of checking was not set to the correct value.

# Question 1

The worst time is 2ɤ + 3δ. This situation illustrates following picture:



Let process p1 sends heartbeat at 0 and immediately dies. The message is delivered in δ – maximum transmission dely. The time between 0 and 1 is (δ- ɛ) where ɛ → 0 is very small. Process p2 checks for heartbeats at 1, 2 ,3. As heartbeat from p1 was delivered at 1+ ɛ process p2 at 2 will not understand that p1 is dead and understands it only at 3. It means after (2ɤ + 3δ- ɛ), but as ɛ is very small we can consider that it is after (2ɤ + 3δ).

The best case is δ. If the node p1 died less than δ before the check it means that the previous its heartbeat was less than (δ+ ɤ) before and was received during that check round so the node will not be detected as dead during this check.

# Question 2

Can you improve the algorithm for PFD, such that it improves the worst case failure detection time?

Yes. Solutions:

**More timers (more computations)**

Keep a specific timer for every neighbor.

upon event < pp2pDeliver | Pi, [HEARTBEAT]> do

cancel timer for Pi;

startTimer (Pi);

upon event < Timeout | Pi > do

detected := detected Ⅴ {Pi};

trigger < crash | Pi >;

By this way, we have a worst case of:

* Pi sends a heartbeat and dies after ε → 0, that arrives to Pj after δ
* Pj cancels the timer and starts a new one, that will trigger and detect Pi after γ + δ

So, worst case of γ + 2δ

**More messages**

A process can broadcast a heartbeat request to the other processes every γ and then wait for 2δ (2 \* max network delay) for the response. If one process does not response in this time, it has crashed.

By this way, we have a worst case of:

* Pi sends a heartbeat request to Pj (and all other nodes), request that arrives after ε → 0
* Pj responses after ζ → 0
* Pi checks for crashes after **2δ**, Pj looks ok
* Pi sends a heartbeat request to Pj (and all other nodes) after γ
* Pi detects the crash of Pj after **2δ**

So, worst case of **2δ + γ + 2δ = γ + 4δ**

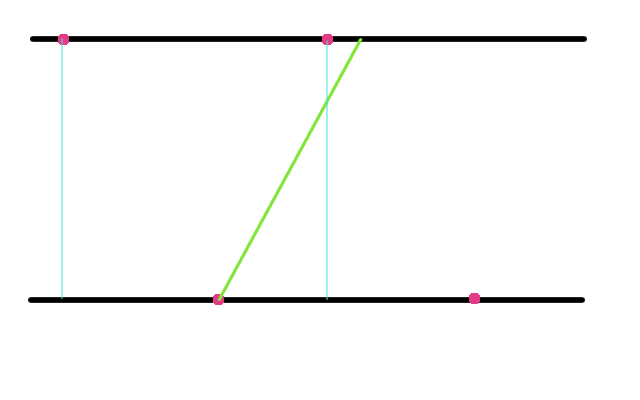
# Question 3

Can you improve the algorithm for PFD, such that it requires a single timer?

If we use a heuristic solution, we can answer yes. In the algorithm that we used for this homework, we can use **only one timer** with period **δ** and select a value for γ, such that γ = ν \* δ, where ν ε Ι. Then we can simply use a counter and two ifs to select when each part of the algorithm will execute.

Else, if we want to use an algorithm that is making use of only one timer, we would have to use the book's algorithm, which we will prove that is wrong in a specific case.

Assume that the TimeDelay that is used in the book's algorithm can be written as:



TimeDelay = γ + δ, where δ is the maximum network delay. If we do not assume that the cycles in every node are synchronized, then there is the chance that the following execution could appear:

Which would obviously cause a false alarm.