Name: Cezar Augusto Contini Bernardi

Student ID: 140139

Obligatory exercise 2

2014-09-15

Task:

- **Imagine an electronic id solution (e.g. login to e-voting).**
- **Think about two (or more) possible race conditions and their effects.**
- **Describe countermeasures for the race conditions you observe.**

Introduction

A race condition can be defined as an anomalous behavior due to unexpected critical dependence on the relative timing of events [1]. It means that any pair of operations in a secure program must still work correctly if arbitrary amounts of another process's code is executed between them [2].

Possible Race Condition 1

If the users A and B want to vote on the candidate X and do it at the same time, the server may receive vote from B while summing vote from A, resulting in two separated sums, causing it to store only the last one executed. By doing so, one vote would be lost.

It is going to happen every time a second user votes, starting by reading the current total number of votes, while another one is in the middle of the sum process (before writing his own vote to the total).

Possible Countermeasure 1

This race condition can be easily avoided by using a mutex. It is a mutual exclusion object that allows multiple threads to synchronise access to a shared resource [1].

A mutex has two states, locked and unlocked. Every time a thread, or process, tries to use a resource that is controlled by a mutex, it will only do so if the mutex is unlocked, if it is not, then the thread will wait until it is to proceed. In the case of multiple threads waiting to use the resource, a set of priorities can be implemented to decide wich one will get it first.

Possible Race Condition 2

If the system is designed in a way that a user can send multiple votes almost instantly, by clicking multiple times on a “vote” button, the system may register more than one vote from the same user. It would happen if there was a verification on witch the user has voted or not before the mentioned sum function above, and this same function would only register a vote after saving it. This way, if a second request gets to the server before it flags the user as “already voted”, the vote would count twice.

Possible Countermeasure 2

The first countermeasure would solve this problem as well. It works because there would never be two processes simultaneously voting.

References

[1] FOLDOC. (2014). Free Online Dictionary of Computing [Online]. URL: http://foldoc.org/. (17.09.2014)

[2] Wheeler, David A. (2014). Avoid Race Conditions [Online]. URL:
http://www.dwheeler.com/secure-class/Secure-Programs-HOWTO/avoid-race.html
(17.09.2014)