Second assignment in Software security

Name: Katrine Borge
Student number: 120919
Class: 12HBISA

Introduction:
What is a race condition?
A race condition is when two or more threads need to use the same resource at the same time, and it becomes a “race” for who gets to use the resource first.

Typically one thread wants to read a resource and another wants to perform an operation or write the resource back to memory. Because there are no way of knowing how the scheduling algorithm will schedual these operations in a multi-threaded environment, bad or illogical things can happen if no countermeasures are taken. [1]

Example of race condition:
We can start with a short and simple example code.

```
1.  if (x =! 0) // Checks if x is anything but 0
2.  {
3.      y = x - 1; // Store x minus one in y.
4.  }
5.  else
6.  {
7.      y = x + 1; // Store x pluss one in y
8.  }
```

If the first thread performs the check at line 1. Then another thread gets to do something inbetween the check (line 1) and the action (line 3 - subtraction) then the number stored in x might be changed allready by another thread. It’s hard to see if you have a race condition or not by just running the code, because race condidions might not run more than one in a thousand times, or even less often. But when it first hits, it can have severe consequences.

Another typical example is when you want to buy tickets, and you get the message that there is still more tickets left. But by the time you actually click “buy”, someone else have already bought your tickets. While a system can be made to work like this on purpose, this often leads to a race condition.

The imaginary game

In this assignment I am going to look at a generic online game with a poll system and a bank transactions system.

Case 1: Preform a poll vote:

Problem:

If two people vote on the same option simultaneously and at the exact same time, a race condition can occur.

Psudocode:

```
1.  vote(option)
2.  {
3.      numVotes = getVotes(option);
4.      numVotes = numVotes +1;
5.      setVotes(numVotes);
6.  }
```

What happends here is that when the first voter chose his option and try to add it, then another process takes over just when the vote is about to get counted. Then when the second voter have had it's vote counted, and the first process is getting back the controll he overwrites the vote from the second voter. This leads to loss in votes.

Solution:

The solution to this is to add a lock to the critical section, so that the process will have controll untill it's done with the task.

Psudocode:

```
1.  vote(option)
2.  {
3.      pthread_mutex_lock(&count_mutex);
4.      numVotes = getVotes(option);
5.      numVotes = numVotes +1;
6.      setVotes(numVotes);
7.      phread_mutex_unlock(&count_mutex);
8.  }
```

Case 2: Bank transaction

Problem:

In this scenario we will look at how race conditions can occur in bank transactions. To perform a bank transaction, the account needs to have enough money stored in it. In this example the balance is checked to see if it actually have enough money to send. The problem occurs when the balance is checked, and another process takes over the controll. This can lead to having more transactions happening without it being enough on the account, and many other outcomes.

Pseudocode:

```
1.    SendMoney(amount, receiver)
2.    {
3.        myBalance = getBalance(me);
4.        receiverBalance = getBalance(receiver);
5.
6.        If (myBalance >= amount)
7.        {
8.            myBalance = myBalance - amount;
9.            receiverBalance = receiverBalance + amount;
10.       }
11.   }
```

Solution:

To solve this we can use mutex to lock in the critical sector that is from point 3. - 10. This will make sure the whole transaction is done before another process can take controll. Could also put a limit to how fast and how many transactions that one can do in a certain amount of time.

Resources:

http://stackoverflow.com/questions/34510/what-is-a-race-condition(16.09.14) [1]
http://support.microsoft.com/kb/317723 (16.09.14)
http://searchstorage.techtarget.com/definition/race-condition (16.09.14)
http://www.dwheeler.com/secure-class/Secure-Programs-HOWTO/avoid-race.html (16.09.14)
http://docs.oracle.com/cd/E19455-01/806-5257/sync-12/index.html (17.09.14)
http://www.w3schools.com/php/php_sessions.asp (17.09.14)
http://www.makelinux.net/books/lkd2/ch08lev1sec1 (17.19.14)