Assigment 2

Tommy André Evensen

120484

12HBISA

Introduction

The Assignment :

« Imagine an electronic ID solution, think about two (or more) possible race conditions and their effects. Describe countermeasures for the race conditions you observe»

Race condition is the behavior of an electronic or software system where the output is dependent on the sequence or timing of other uncontrollable events. Its like a bug when events do not happen in the order intended. When programming with mutible threads which access the same resources, a race condition may happen.

```
1  if (x == 5)
2  {
3      y = x * 2;
4  }
```

Problems often occur when one thread does a «check-then-act» and another does something to the value between the «check» and the act. Example over the «if» statement is the check, and the y=x*2 is the act of this program.

In this little code, the expected output is y = 10, but if another thread changed x in between the check and the act. The resoult could be anything. We have no idea knowing the actually resoult, it could be 10 or it could be something compleatly diffrent.

First Case - Voting

In this poorly written code, we are at risk of some votes not getting counted. That could happen as some votes may be registrert at the same time and therefore setting wrong values. And we know that every votes counts so we cant be having any of that now, can we?

```
1  Void RegistrerVote(user user, party party)
2      int user_nr = user_nr();
3      int party_nr = Party_nr();
4
5      if (user_nr != har_stemt)
6          {
7          put_vote_in_box(party_nr);
8          Har_stemt(user_nr) = True;
9          }
10     else {
11     cout << "du har stemt!!";
12         Loggout();
13     }
```

(!= on line 5 is just to mark that user have not voted)

The solution to this problem is using binary semaphores. Semaphores is a variable or abstract data type that is used to controlling access to a common resource in a paralell programming or a multi user enviroment. In The exemple under we have added the semaphores.

```
1  Public binarySemaphore sem;
2
3  Void RegistrerVote(user user, party party)
4      int user_nr = user_nr();
5      int party_nr = Party_nr();
6
7      if (user_nr != har_stemt)
8          {
9          Sem.lock();
10         put_vote_in_box(party_nr);
11         Har_stemt(user_nr) = True;
12         Sem.unlcok();
13         }
14     else {
15     cout << "du har stemt!!";
16         Loggout();
17     }
```

Now the voteing process is protected in the critical region as u se the lock at line9 and unlock at line 12. We should now get all the votes, and the democracy is restored.

Case 2 - Comments

In this scenario, we pictures us a comment section on a website. The comments have their own id's and the commenter have unique user id.

Lets pretend the program works like this:

.....
1. comment = new comment(commentNR, UserID);
2. nr++;
3. Add_comment(comment);
.....

Now again, as the previous example, if two people make a comment at the same time, and the thread of the first one runs line line 1, and the second thread runs line 1 before the first gets to nr++. Both comments will now have the same comment ID, and the nr variable will increase from 1 to 3. This can be a problem when it comes for a administrator to find, delete or modify a comment. I would imagine a user may delete both if he wants tries to delete his own.

Again I would use a mutex or a binary semaphore to lock the critical region.

Sources:

1. M. Dowd, J. McDonald, and J. SCHUH, The art of software security assesment identifying and preventing software vulnerabilities.

2. Race conditions examples. http://infohost.nmt.edu/~eweiss/222_book/222_book/0201433079/ch08lev1sec9.html#ch08fig13

3. Semaphores secondary source http://en.wikipedia.org/wiki/Semaphore_(programming)