Assignment 2

Kim Roger Torgersen
120444

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.

What is a Race Condition?

A race condition or race hazard is the behavior of an electronic or software system where the output is dependent on the sequence or timing of other uncontrollable events. It becomes a bug when events do not happen in the order the programmer intended. The term originates with the idea of two signals racing each other to influence the output first. Race conditions can occur in electronics systems, especially logic circuits, and in computer software, especially multithreaded or distributed programs.(1)

The system:

An online bank system in this situation used to manage a corporate bank account where several persons tries to pay bills from different workstations.

Problem #1:

Two users simultaneously tries to pay a bill from their department. This could result in that while user 1 is processing its payment the system gives control to user 2 which then proceeds to pay his bill. This can result in one of the bills not coming through, or that the amount paid is not correct as the users are working on the same variables. It could also be that the balance on the account is not enough for both, but since the balance was enough for that user when the bill was added, the system will not notify the user, but the banksystem would stop the transaction because of insufficient funds.

Pseudo-code for this process could be something like:

```
1. Verify user
2. Get recipients information
3. Get bill amount
4. If(Current balance > bill amount)
5.         Process_Transaction(recipient, bill_info)
6. Else
7.         Notify user
```

The critical sector here would be line 4-7(2 p115)

Problem #2:

The same corporation wants a receipt of this months transactions. While the systems work on processing the request of printing this, a user adds a last minute bill. This may result in the print being different from what the screen showed for the user who printed.

Pseudo-code for this process could be something like:

```
1. Verify user
2. If(uncompleted transactions exists)
3.         Process_Transaction(uncomplete transactions)
4. Get current months transactions
5. If(User requests print)
6.         Print_transactions(current month)
```

The critical sector here would be line 4.(2 p115)

The solution:

The solution for both problems would be to implement the use of mutex or semaphores to lock the critical sectors from being changed by several users at the same time.

Code example:

```
1. #define N 100;
2. Typedef int semaphore;
3. semaphore mutex = 1;
4. semaphore empty = N;
5. Semaphore full = 0;
6.  //Getting/Setting variables
7. down(&empty);
8. down(&mutex);
9. //Critical sector
10. up(&mutex);
11. up(&full);
```

Code example above is taken from reference 2 page 128.

Using this would prevent multiple users changing the same value while the system processes it and therefor prevent that what the system uses for variables is a mix between what several users input.

References

1. Wikipedia. Race Condition. Wikipedia; 2014 [sitert 16.09]; Tilgjengelig fra: http://en.wikipedia.org/wiki/Race_condition.
2. Tanenbaum AS. Modern Operating Systems; Third Edition2009.