Gjøvik University College

Software Security

Race Conditions

Assignment #2

Student:
Tommy B. Ingdal

September 17, 2014

Contents

1 Introduction

Race Condition can be defined as "*Anomalous behavior due to unexpected critical dependence on the relative timing of events*" [2].
When doing multithreaded programming where multiple threads share the same data, a condition called *race condition* may occur. If the programmer don't define a critical section and use some sort of locking mechanism to protect the shared data, unexpected behaviour can cause serious problems.

In this paper we will look into two examples on how a race condition can happen and how we can implement some sort of locking mechanism to mitigate this problem.

2 Examples

2.1 E-voting

A race condition may occur when two persons vote for the same option at the same time.
This is demonstrated with the following code snippet:

```
void addVote(someOption)
{
    vote = readVotes(someOption);
    vote++;
    updateVotes(vote);
}
```

Listing 1: Race condition in e-voting

Person #1 vote for a specific party and tries to increment the total number of votes. But before this happens the thread is preempted [5] and another thread starts running. The second thread (person) gets his vote counted and thread #1 gets back the control.
This thread now writes to the vote variable and overwrites person #2's vote. Only one vote is counted.

2.2 User Login

When you log into an application you usually enter your username and password. Then the application checks the database to see if your username and password match a stored record. If that's the case a session is set and you're logged into the system.
But what happens if the running thread is preempted by the processor and another thread starts running between the check of credentials and the actual login?

A race condition can occur when a thread performs a *check-then-act* [6] and the processor let's another thread manipulate some shared data between the *check* and *act*.
The following simple piece of code demonstrate a *check-then-act*:

```
1 if (userCred() == storedUserCred()) // The "Check"
2 {
3     user = userCred(); // The "Act"
4     loginUser(user);
5 }
```

Listing 2: Possible race condition (check-then-act)

1. The user enters a username and password and this is then checked against a database.
3. & 4. The username and password match a record in the database and the user is logged into the system.

A race condition may occur and actually log in a different user than expected. If the thread performing the check is preempted, a different thread can possible manipulate the data stored in the variable *user*, and if the control then is sent back to the original thread, a different user may be logged in.
This problem/weakness is called TOCTOU (Time-of-check Time-of-use). [3]

3 Avoid Race Conditions

In order to mitigate the problem with race conditions one should implement a locking mechanism which prevent multiple threads from accessing a shared resource.

By using either a semaphore or mutex this can be achieved by “defining“ a critical section and locking the shared resource.

If we edit the previous example with user login, we can implement a mechanism which mitigates this problem:

```
\\ Lock this section and deny access to the variable user
if (userCred() == storedUserCred())
{
    user = userCred();
    loginUser(user);
}
\\ Release the lock
```

Listing 3: Avoid race condition with a lock

Possible locking mechanisms includes a Mutex [4] or Semaphores [1].

4 Summary

If the programmer don't implement a locking mechanism multiple threads can possible access and manipulate shared data.This can then lead to unexpected behaviour and be really difficult to debug and fix.
To avoid this problem one should “define“ a critical section and use either a semaphore or mutex. By doing this the shared resource can only be accessed by one thread at a time, making the application/code thread-safe.

References

[1] Allen B. Downey. The Little Book of Semaphores. http://greenteapress.com/semaphores/. [Online; accessed 17-September-2014].

[2] dwheeler.com. 7.11. Avoid Race Conditions. http://www.dwheeler.com/secure-class/Secure-Programs-HOWTO/avoid-race.html. [Online; accessed 17-September-2014].

[3] Common Weakness Enumeration. CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition. http://cwe.mitre.org/data/definitions/367.html. [Online; accessed 17-September-2014].

[4] Microsoft Developer Network. Mutexes. `http://msdn.microsoft.com/en-us/library/hw29w7t1%28v=vs.110%29.aspx`. [Online; accessed 17-September-2014].

[5] University Of Illinois Department of Computer Science. CPU Scheduling. `http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html`. [Online; accessed 17-September-2014].

[6] Stack Overflow. What is a race condition? `http://stackoverflow.com/questions/34510/what-is-a-race-condition`. [Online; accessed 17-September-2014].