9. Übungsblatt - C++

Henrik Gerdes, Manuel Eversmeyer
D10

14. Januar 2019

```
Eine Race Condition ist eine Situation, in der mehrere Prozesse
konkurrierend auf gemeinsame Daten zugreifen, hier square_sum,
und mindestenseiner diese manipuliert. Der letztendliche Wert
der gemeinsamen Daten hängt davon ab, in welcher Reihenfolge
die Prozesse darauf zugreifen. Das Ergebnis ist also nicht
vorhersagbar und kann bei überlappenden Zugriffen falsch sein.

Der Ausdruck "square_sum += x * x" ist nur eine verkürtzte
Schreibweise für square_sum = square_sum + x * x. Somit kommen
hier scheibende und lesende Zugriffe vor.

Beispielhafter Ablauf im Fehlerfall:
Thread 1 liest square_sum                           square_sum = 0
Thread 2 liest square_sum                           square_sum = 0
Thread 1 berechnet und schreibt square_sum + 1 * 1   square_sum = 1
Thread 2 berechnet und schreibt square_sum + 2 * 2   square_sum = 4

Die Berechnung von Thread 1 wurde Überschieben (Lost Update)
```

Aufgabe10.1/Aufgabe1.txt

```cpp
#include <iostream>
#include <vector>
#include <thread>
#include <functional>
#include <mutex>

using namespace std;

//int square_sum = 0;
std::mutex mtx;

void pow2(int& square_sum, int x)
{
    mtx.lock();
    square_sum += x * x;
    mtx.unlock();
}

int main(int argc, char const *argv[])
{
    int zero(0);
    auto square_sum = std::ref(zero);

    vector<thread> threads;
    for (int i = 1; i <= 20; i++)
    {
        threads.push_back(thread(&pow2, square_sum, i));
    }

    vector<thread>::iterator it;
    for (it = threads.begin(); it != threads.end(); it++)
    {
       (*it).join();
    }
    cout << "Sum auf squares up to 20 is = " << square_sum << endl;
    return 0;
}
```

Aufgabe10.1/thread1.cpp

```cpp
#include <iostream>
#include <vector>
#include <thread>
#include <functional>
#include <atomic>

using namespace std;

atomic<int> square_sum(0);

int pow2(int x)
{
    int s;
    s = square_sum.load(memory_order_relaxed);
    square_sum.store(s + x * x, memory_order_relaxed);

    return square_sum.load(memory_order_relaxed);
};

int main(int argc, char const *argv[])
{
    std::atomic<int> square_sum(0);

    vector<thread> threads;
    for (int i = 1; i <= 20; i++)
    {
        threads.push_back(thread(pow2, i));
    }

    vector<thread>::iterator it;
    for (it = threads.begin(); it != threads.end(); it++)
    {
       it->join();
    }
    cout << "Sum auf squares up to 20 is = " << pow2(0) << endl;
    return 0;
}
```

Aufgabe10.1/thread2.cpp

```cpp
#include <iostream>
#include <vector>
#include <thread>
#include <functional>
#include <future>
#include <chrono>

using namespace std;

//int square_sum = 0;

int pow2(int x)
{
    return (x * x);
}

int main(int argc, char const *argv[])
{
    //auto square_sum = std::ref(int zero(0));
    int square_sum = 0;

    vector<std::future<int>> threads;
    for (int i = 1; i <= 20; i++)
    {
        threads.push_back(std::async(&pow2, i));
    }

    // vector<std::future<int>>::iterator it;
    // for (it = threads.begin(); it != threads.end(); it++)
    // {
    //     square_sum += it.get();
    // }

    for(auto &e : threads) {
        square_sum += e.get();
    }
    cout << "Sum auf squares up to 20 is = " << square_sum << endl;
    return 0;
}
```

Aufgabe10.1/thread3.cpp

```cmake
##################################
# The name of out project
##################################
project(CPP18_ASTEROIDS)

...

set( CMAKE_CXX_FLAGS         "-pthread -O3 -Wall" )
set( CMAKE_CXX_FLAGS_RELEASE "-O3 -msse3 -Wno-deprecated-declarations -Wno-unused -Wcpp" )
set( CMAKE_CXX_FLAGS_DEBUG   "-g -Wall" )

...

set(ASTEROID_SOURCES
    Main.cpp
    #io/PLYIO.cpp #old ply loader
    io/PLYTraits.cpp
    io/ReadPPM.cpp
    io/ReadTGA.cpp
    io/ReadJPG.cpp
    io/ReadPLY.cpp
    io/Read3DS.cpp
    io/ReadOBJ.cpp
    io/TextureFactory.cpp
    io/TriangleMeshFactory.cpp
    math/Matrix.cpp
    math/Quaternion.cpp
    math/Randomizer.cpp
    rendering/Rectangle.cpp
    rendering/Renderable2D.cpp
    rendering/Renderable3D.cpp
    rendering/Circle.cpp
    rendering/Sphere.cpp
    rendering/Texture.cpp
    rendering/TriangleMesh.cpp
    rendering/TexturedMesh.cpp
    rendering/Skybox.cpp
    rendering/SpaceCraft.cpp
    rendering/Bullet.cpp
    rendering/Asteroid.cpp
    rendering/AsteroidField.cpp
    util/Util.cpp
    view/MainWindow.cpp
    view/Camera.cpp
)

...

add_executable(Counter-Mutex Aufgabe10.1/thread1.cpp)
add_executable(Counter-Atomic Aufgabe10.1/thread2.cpp)
add_executable(Counter-Tasks Aufgabe10.1/thread3.cpp)
add_executable(asteroids ${ASTEROID_SOURCES} ${C3DSREADER_SOURCES})
```

CMakeLists–Kopie.txt

```
/*
 *  Bullet.hpp
 *
 *  Created on: Jan. 06 2019
 *      Author: Thomas Wiemann
 *
 *  Copyright (c) 2019 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#ifndef BULLET_HPP_
#define BULLET_HPP_

#include <memory>
#include <thread>
#include "../math/Vector.hpp"
```

```cpp
#include "Sphere.hpp"

namespace asteroids
{

/**
 * @brief Renders a Bullet
 */
class Bullet
{

public:

    using Ptr = std::shared_ptr<Bullet>;

    /**
     * @brief Contructor. Build a bullet on the given Fighter's
     *                       position. The bullet will move on the
     *                       given axis.
     * @param   fighter_position   Position of the fighter that shoots this
         bullet
     * @param   fighter_axis   Axis the bullet will move on
     */
    Bullet(const Vector3f& fighter_position, const Vector3f fighter_axis);

    ~Bullet();

    /**
     * @brief Moves the bullet until it's lifetime is over.
     */
    void run();

    /**
     * @brief Starts bullet movement
     */
    void start();


    /*
     * @brief Stops bullet movement
     */
    void stop();

    /**
     * @brief Renders the bullet via glutSolidSphere.
     */
    void render();

    /**
     * @brief Returns the status of this bullet.
     * @return false, if the bullet's lifetime is over and true otherwise
     */
    inline bool isAlive()
    {
        return m_alive;
    };

private:

    //Liftime
    const static int m_lifetime = 900;

    // Tells if bullit is vlaid
    bool m_alive;

    //Axis
    Vector3f m_flightAxis;

    //Spaceship position
    Vector3f m_fighter_position;

    //Tread for bullet
    thread m_thread;

    /// Sphere objet to render the bullet
    Sphere m_sphere;
};

} // namespace asteroids


#endif /* BULLET_HPP_ */
```

rendering/Bullet.hpp

```
/**
 * @file Bullet.cpp
 * @author Henrik Gerdes (hegerdes)
 * @brief
 * @version 0.1
 * @date 2019-01-13
 *
 * @copyright Copyright (c) 2019
 *
 */

#include "Bullet.hpp"

namespace asteroids
{

Bullet::Bullet(const Vector3f& fighter_position, const Vector3f
    fighter_axis):
    m_alive(true), m_flightAxis(fighter_axis), m_fighter_position(
        fighter_position),
    m_sphere(fighter_position,10)
    {
        //m_sphere = fighter_position;
    }

void Bullet::render()
{
    m_sphere.render();
}

void Bullet::run()
{

    int i = 0;
    // Modify the bullet's position until the lifetime is over
    while(i < Bullet::m_lifetime){

        //m_sphere.move(Renderable3D::ACCEL, 5);
        m_sphere.setPosition(m_sphere.getPosition() + m_flightAxis);

        i++;
        std::this_thread::sleep_for(std::chrono::microseconds(1000));
    }
    m_alive = false;

}

void Bullet::start()
{
    m_thread = std::thread(&Bullet::run, this);
}

void Bullet::stop()
{
    m_thread.join();
}

Bullet::~Bullet()
{
    //stop();
}


}//asteroids
```

rendering/Bullet.cpp

```
/*
 *  SpaceCraft.hpp
 *
 *  Created on: Jan. 06 2019
 *      Author: Thomas Wiemann
 *
 *  Copyright (c) 2019 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#ifndef __SpaceCraft_HPP__
#define __SpaceCraft_HPP__

// Local includes
#include "TriangleMesh.hpp"
#include "Bullet.hpp"

// SDL includes
#include <SDL2/SDL.h>

```

```cpp
// Standard includes
#include <list>

namespace asteroids
{
/**
 * @brief A class to render a cicle to the screen
 *
 */
class SpaceCraft
{
public:
    /**
     * @brief Construct a new SpaceCraft object
     *
     * @param filename the filename of the Trianglemesh
     * @param position the start position
     * @param movespeed movespeed for the key handling
     * @param rotatespeed rotatespeed for the key handling
     */
    SpaceCraft(const std::string& filename, const Vector3f& position, float
        movespeed, float rotatespeed);

    /// dtor
    virtual ~SpaceCraft();

    /**
     * @brief moves and rotates the mesh
     *
     * @param keyStates the SDL Keyinput
     */
    void handleKeyInput(const Uint8* keyStates);

    /// renders the SpaceCraft
    void render();

    /**
     * @brief check if the SpaceCraft has a valid mesh
     *
     * @return true if mesh is valid
     * @return false if mesh is not valid
     */
    bool hasMesh() const;

    /// Adds a bullt to the ship's list of active bullets
    void shoot();

private:

    /// Internal triangle mesh
    TriangleMesh::Ptr m_mesh;

    /// Move speed
    float m_movespeed;

    /// Turning speed
    float m_rotatespeed;

    /// List of active bullets
    std::list<Bullet::Ptr> m_bullets;
};

} // namespace asteroids

#endif
```

rendering/SpaceCraft.hpp

```
/*
 *  SpaceCraft.cpp
 *
 *  Created on: Nov. 04 2018
 *      Author: Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#include "SpaceCraft.hpp"
#include "Bullet.hpp"
#include <iterator>
#include <algorithm>
#include "io/TriangleMeshFactory.hpp"

namespace asteroids
{

SpaceCraft::SpaceCraft(const std::string &filename, const Vector3f&
    position, float movespeed, float rotatespeed)
    : m_movespeed(movespeed), m_rotatespeed(rotatespeed)
{
    m_mesh = TriangleMeshFactory::instance().getMesh(filename);
    if(m_mesh)
    {
        m_mesh->setPosition(position);
    }
}

void SpaceCraft::handleKeyInput(const Uint8* keyStates)
{
    if (keyStates[SDL_SCANCODE_UP])
    {
        m_mesh->rotate(TriangleMesh::YAW, m_rotatespeed);
    }
    if (keyStates[SDL_SCANCODE_DOWN])
    {
        m_mesh->rotate(TriangleMesh::YAW, -m_rotatespeed);
    }
    if (keyStates[SDL_SCANCODE_LEFT])
    {
        m_mesh->rotate(TriangleMesh::ROLL, m_rotatespeed);
    }
    if (keyStates[SDL_SCANCODE_RIGHT])
    {
```

```cpp
        m_mesh->rotate(TriangleMesh::ROLL, -m_rotatespeed);
    }
    if (keyStates[SDL_SCANCODE_W])
    {
        m_mesh->move(TriangleMesh::ACCEL, -m_movespeed);
    }
    if (keyStates[SDL_SCANCODE_S])
    {
        m_mesh->move(TriangleMesh::ACCEL, m_movespeed);
    }
    if (keyStates[SDL_SCANCODE_A])
    {
        m_mesh->move(TriangleMesh::STRAFE, -m_movespeed);
    }
    if (keyStates[SDL_SCANCODE_D])
    {
        m_mesh->move(TriangleMesh::STRAFE, m_movespeed);
    }
    if (keyStates[SDL_SCANCODE_SPACE])
    {
        shoot();
    }
}

void SpaceCraft::render()
{
    m_mesh->render();

    // for(auto const &b:m_bullets)
    // {
    //     b->render();
    // }

    // auto it = remove_if(m_bullets.begin(), m_bullets.end(), [](Bullet::Ptr b)->bool{
    //     return !(b->isAlive());
    // });

    // m_bullets.erase(it,m_bullets.end());

    for(std::list<Bullet::Ptr>::iterator it = m_bullets.begin(); it != m_bullets.end(); ++it)
    {
        if(it->get()->isAlive())
        {
            it->get()->render();
        }
        else
        {
            //it = m_bullets.erase(it, m_bullets.end());
        }
    }
}

bool SpaceCraft::hasMesh() const
{
    return m_mesh != nullptr;
}
```

```

void SpaceCraft::shoot()
{
    Bullet::Ptr b(new Bullet(m_mesh->getPosition(),m_mesh->getxAxis() *
        -1.0));

    b->start();

    m_bullets.push_back(b);
}

SpaceCraft::~SpaceCraft()
{
    for(auto th:m_bullets)
    {
        th->stop();
    }

}

} // namespace asteroids
```

rendering/SpaceCraft.cpp