

PROGRAMMIEREN II

DHBW Stuttgart Campus Horb INF2017

LETZTE AUFGABE

- Auslesen der folgenden Informationen aus dem Diablo 2 Charakter im Repository
- Das Dateiformat ist in der eingecheckten PDF beschrieben
- Finde die folgenden Informationen über den Helden hinaus und speichere sie in einer entsprechenden Struktur / Klasse
 - Name
 - Status (lebendig, hardcore)
 - Titel
 - Klasse
 - Level
- Tipp: Mit ifstream::ignore können bytes übersprungen werden

AGENDA

- Threads
- Synchronisierung

PROGRAMMIER AUFGABE ZUM WARMWERDEN

- Lies die Datei bin_out in den Speicher und knacke den Verschlüsselungscode
- Die Datei ist wie folgt abgelegt:
 - int l 6_t für die Größe der folgenden Nachricht
 - · Dann ein Block in der Größe mit der Nachricht
 - Die Nachricht ist in verkehrter Reihenfolge und jeder Char + I gerechnet

THREAD

- Konstrukt für Nebenläufigkeit
- · Gerne auch als leichtgewichtiger Prozess bezeichnet
- Es gibt unterschiedliche Arten von Threads
 - Kernel Threads
 - User Threads auch Green threads genannt

KERNELTHREAD

- Ein Thread ist ein sequentieller Bearbeitungsablauf innerhalb eines Prozesses
- Er teilt sich mit anderen Threads die sogenannten Betriebsmittel:
 - Codesegment
 - Datensegment
 - · Dateidescriptoren (häufig im Datensegment)

KERNELTHREAD

- Jeder Thread besitzt eine Reihe von eigenen Resourcen, die im sogenannten Threadkontext zusammengefasst werden:
 - · Unabhängiger Registersatz inkl. Instruction Pointer
 - Einen eigenen Stack (jedoch geteilten Heap!)
 - Eventuell: Thread Local Storage

KONFLIKTE

- Da der Heap geteilt ist und Zeiger eventuell in mehreren Threads aktiv sind kann es zu Ressourcenkonflikten kommen
- —> Es werden Synchronisationsmechanismen benötigt (siehe VL ParaProg)

C++ UND THREADS

- Bis C++11 keine Standardisierte Thread Library
- Es musste direkt auf der entsprechenden API gearbeitet werden (Windows Threads, POSIX)
- Boost::thread als Zwischenlösung
- Dieser wurde in C++11 standardisiert

THREAD EXAMPLE

```
// thread example
#include <iostream>
                          // std::cout
#include <thread>
                          // std::thread
void foo()
    do stuff -> Heavy Computing...
void bar(int x)
  // do stuff -> Heavier Computing...
int main()
                            // spawn new thread that calls foo()
  std::thread first (foo);
  std::thread second (bar,0);
                              // spawn new thread that calls bar(0)
  std::cout << "main, foo and bar now execute concurrently...\n";</pre>
  // synchronize threads:
  first.join();
                               // pauses until first finishes
  second.join();
                               // pauses until second finishes
  std::cout << "foo and bar completed.\n";</pre>
  return 0;
```

WAS SEHEN WIR IM ERSTEN BEISPIEL?

- std::thread als Implementierung von Threads
- Der Construktor nimmt eine Funktion / einen
 Funktionspointer und die zu übergebenden Parameter an
- Thread wird mit dem Constructor direkt gestartet
- Mit Join wird ein klassischen Zusammenführen der Threads ermöglicht

WAS MACHE ICH WENN ICH EINEN THREAD ALS MEMBER NUTZEN MÖCHTE?

std::thread second (bar,0); // spawn new thread that calls bar(0)



DIE LÖSUNG!

- Pointer!
- Die noch bessere Lösung?
 - Smartpointer

MUTEX

```
#include <iostream>
#include <chrono>
#include <thread>
#include <mutex>
#include <map>
#include <string>
std::map<std::string, std::string> g_pages;
std::mutex g_pages_mutex;
void save_page(const std::string &url)
    std::this_thread::sleep_for(std::chrono::seconds(2));
    std::string result = "fake content";
    g_pages_mutex.lock();
g_pages[url] = result;
    g_pages_mutex.unlock();
int main()
    std::thread t1(save_page, "http://foo");
std::thread t2(save_page, "http://bar");
    t1.join();
    t2.join();
    g_pages_mutex.lock();
    for (const auto &pair : g_pages) {
        std::cout << pair.first << " => " << pair.second << '\n';</pre>
    g_pages_mutex.unlock();
```

PROBLEME MIT MUTEX?

```
#include <iostream>
#include <chrono>
#include <thread>
#include <mutex>
#include <map>
#include <string>
std::map<std::string, std::string> g_pages;
std::mutex g_pages_mutex;
void save_page(const std::string &url)
    std::this_thread::sleep_for(std::chrono::seconds(2));
    std::string result = "fake content";
    g_pages_mutex.lock();
g_pages[url] = result;
    g pages mutex.unlock();
int main()
    std::thread t1(save_page, "http://foo");
std::thread t2(save_page, "http://bar");
    t1.join();
    t2.join();
    g_pages_mutex.lock();
    for (const auto &pair : g_pages) {
        std::cout << pair.first << " => " << pair.second << '\n';</pre>
    g_pages_mutex.unlock();
```

PROBLEM LÖSUNG FÜR EXCEPTIONS:

```
#include <thread>
#include <mutex>
#include <iostre<mark>am></mark>
int g_i = 0;
std::mutex g_i_mutex; // protects g_i
void safe_increment()
    std::lock_guard<std::mutex> lock(g_i_mutex);
    ++g_i;
    std::cout << std::this_thread::get_id() << ": " << g_i << '\n';
    // g_i mutex is automatically released when lock
    // goes out of scope
int main()
    std::cout << "main: " << g_i << '\n';
    std::thread t1(safe_increment);
    std::thread t2(safe_increment);
    t1.join();
    t2.join();
    std::cout << "main: " << g_i << '\n';
```

LESEN != SCHREIBEN

```
#include <iostream>
#include <mutex> // For std::unique_lock
#include <shared_mutex>
#include <thread>
class ThreadSafeCounter
 public:
   ThreadSafeCounter() = default;
   // Multiple threads/readers can read the counter's value at the same time.
   unsigned int get() const
        std::shared_lock<std::shared_mutex> lock(mutex_);
       return value :
   // Only one thread/writer can increment/write the counter's value.
   void increment()
       std::unique_lock<std::shared_mutex> lock(mutex_);
       value ++;
   // Only one thread/writer can reset/write the counter's value.
   void reset()
        std::unique_lock<std::shared_mutex> lock(mutex_);
       value = 0:
  private:
   mutable std::shared_mutex mutex_;
   unsigned int value = 0;
```

PROGRAMMIER AUFGABE II

- Nehme die Aufgabe von vorhin und erzeuge mit gen_bin.cpp mehrere solcher Binär-Dateien
- Schreibe ein Programm, dass eine Liste von Dateien als Start-Parameter bekommt
- · Jede dieser Dateien soll in einem eigenen Thread entschlüsseln
- Am Ende soll auf der Kommandozeile ausgegeben werden, welche Datei mit welcher Größe und welcher Anzahl Nachrichten entschlüsselt wurden sowie wie viele Sekunden dies benötigt hat