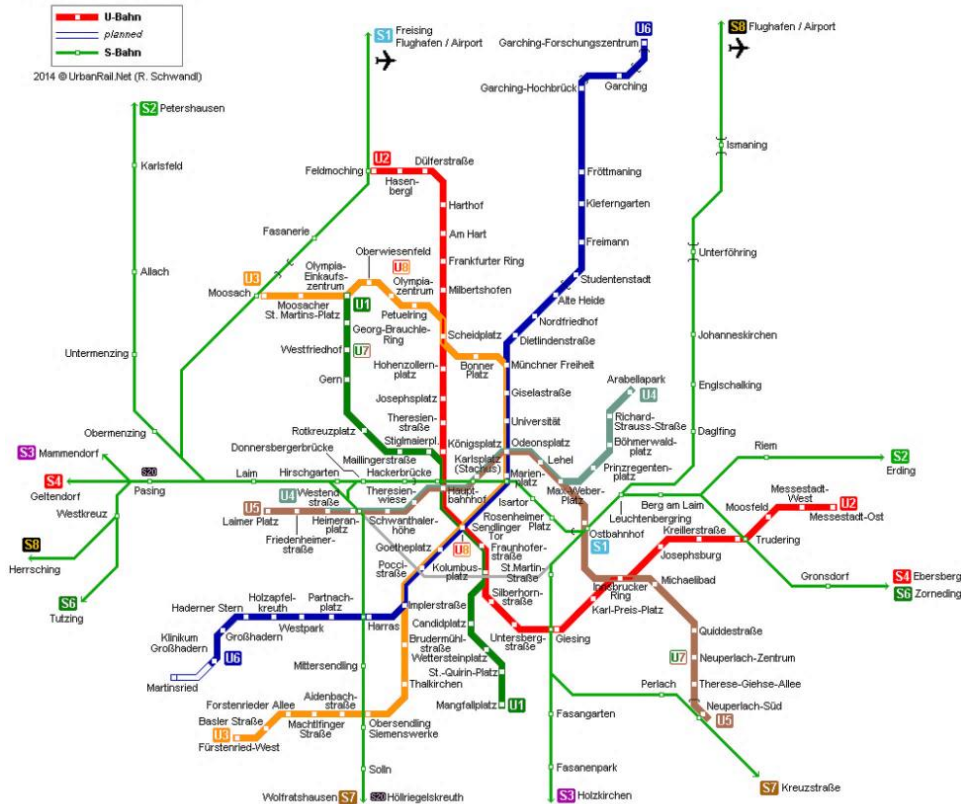


Multithreading, richtig gemacht?

Rainer Grimm

Überblick

MÜNCHEN · MUNICH



- Threads
- Geteilte Variablen
- Thread lokale Daten
- Bedingungsvariablen
- Promise und Future
- Speichermodell

Threads

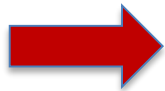


- Benötigt ein Arbeitspaket und startet sofort
- Der Erzeuger des Threads muss sich um sein Kind kümmern
 - auf sein Kind warten
 - von seinem Kind trennen
- Nimmt die Daten per Copy oder Referenz an

Probleme

```
#include <iostream>
#include <thread>

int main(){
    thread t( []{cout << this_thread::get_id();} );
}
```

A screenshot of a terminal window titled "bin : bash - Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Lesezeichen", "Einstellungen", and "Hilfe". The terminal text shows a user named "rainer" at a prompt "rainer@icho:~>" typing "threadForgetJoin". The output is "terminate called without an active exception" followed by "Aborted" on the next line. The prompt "rainer@icho:~>" is shown again on the next line. The terminal has a status bar at the bottom with "bin : bash" and a red 'x' icon on the right.

```
bin : bash - Konsole
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
rainer@icho:~>threadForgetJoin
terminate called without an active exception
Aborted
rainer@icho:~>
```

join oder detach

```
#include <iostream>
#include <thread>

int main(){
    thread t( []{cout << this_thread::get_id();} );
    t.join();
    // t.detach();
}
```

A screenshot of a terminal window titled "bin : bash - Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Lesezeichen", "Einstellungen", and "Hilfe". The terminal content shows the command "rainer@icho:~>threadJoin" followed by the output "140727045601024" and a new prompt "rainer@icho:~>". The terminal has a yellow background and a grey status bar at the bottom showing "bin : bash".

```
bin : bash - Konsole
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
rainer@icho:~>threadJoin
140727045601024
rainer@icho:~>
```

Probleme?

```
#include <iostream>
#include <thread>

int main(){
    thread t([]{cout << this_thread::get_id();});
    thread t2([]{cout << this_thread::get_id();});

    t= std::move(t2);

    t.join();
    t2.join();
}
```



```
bin : bash - Konsole <2>
File Edit View Bookmarks Settings Help
rainer@linux:~>threadMove
terminate called without an active exception
Aborted
rainer@linux:~>
```

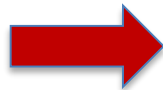
join oder detach

```
#include <iostream>
#include <thread>

int main(){
    thread t([]{std::cout << this_thread::get_id();});
    thread t2([]{std::cout << this_thread::get_id();});

    t.join();
    t = std::move(t2);
    t.join();

    cout << "t2.joinable(): " << t2.joinable();
}
```

A screenshot of a terminal window titled "bin : bash - Konsole <2>". The window has a menu bar with "File", "Edit", "View", "Bookmarks", "Settings", and "Help". The output shows the program execution: "rainer@linux: ~> threadMove", followed by two thread IDs "139829891147520" and "139829882754816", and finally "t2.joinable(): false". The prompt "rainer@linux: ~>" is visible at the bottom. The terminal has a taskbar at the bottom with "bin : bash" and a close button.

```
bin : bash - Konsole <2>
File Edit View Bookmarks Settings Help
rainer@linux: ~> threadMove
139829891147520
139829882754816

t2.joinable(): false
rainer@linux: ~>
```

scoped_thread

```
class scoped_thread{
    std::thread t;
public:
    explicit scoped_thread(std::thread t_):t(std::move(t_)){
        if ( !t.joinable() ) throw std::logic_error("No thread");
    }
    ~scoped_thread(){
        t.join();
    }
    scoped_thread(scoped_thread const&)= delete;
    scoped_thread& operator=(scoped_thread const &)= delete;
};

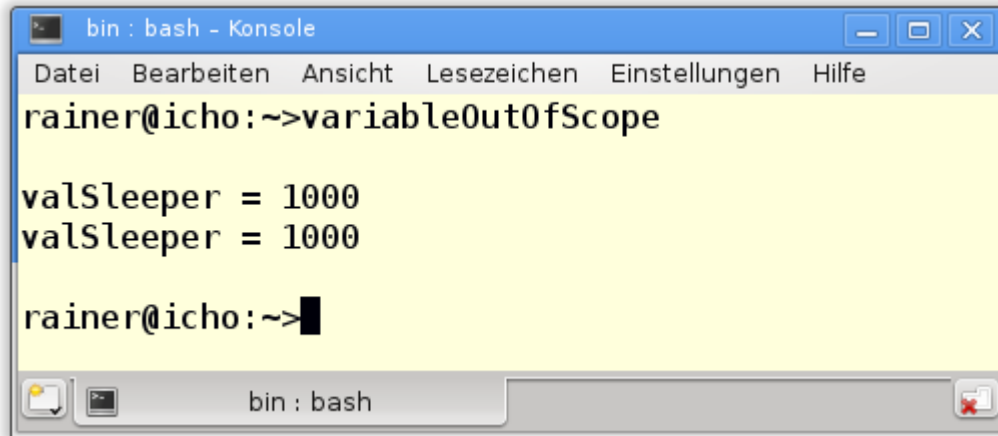
int main(){
    scoped_thread t(std::thread([]{std::cout << this_thread::get_id();}));
}
```

Probleme?

```
struct Sleeper{
    Sleeper(int& i_):i{i_}{};
    void operator() (int k){
        for (unsigned int j= 0; j <= 5; ++j){
            this_thread::sleep_for(chrono::milliseconds(100));
            i += k;
        }
    }
private:
    int& i;
};

int main(){
    int valSleeper= 1000;
    cout << "valSleeper = " << valSleeper << endl;
    thread t(Sleeper(valSleeper),5);
    t.detach();
    cout << "valSleeper = " << valSleeper << endl;
}
```

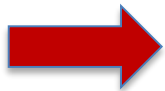
Auswirkungen und Lösungen



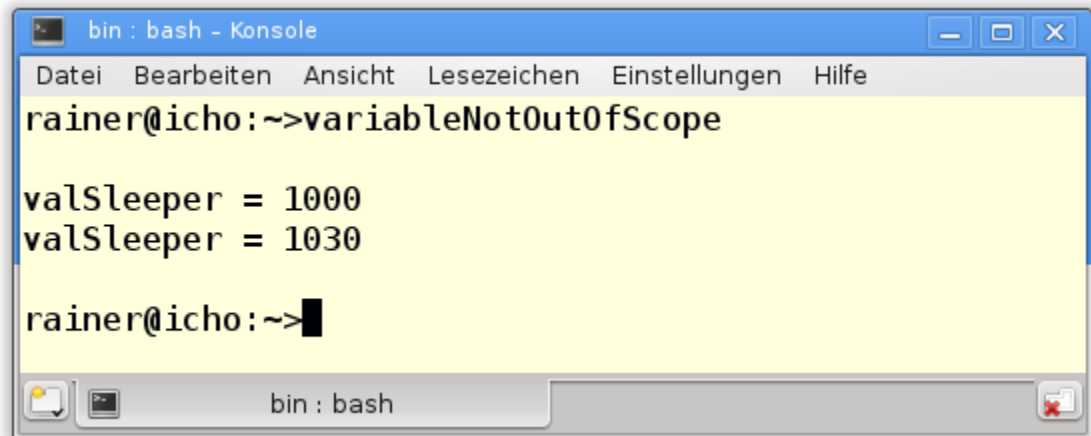
```
bin : bash - Konsole
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
rainer@icho:~>variableOutOfScope

valSleeper = 1000
valSleeper = 1000

rainer@icho:~>
```



```
thread t(Sleeper(valSleeper),5);
t.join(); // instead of t.detach()
```



```
bin : bash - Konsole
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
rainer@icho:~>variableNotOutOfScope

valSleeper = 1000
valSleeper = 1030

rainer@icho:~>
```

Geteilte Variablen



<http://www.robinage.com/>

- Müssen vom geteilten Zugriff geschützt werden
- Atomare Variablen
- Mutexe

Probleme?

```
struct Worker{
    Worker(string n):name(n){};
    void operator() (){
        for (int i= 1; i <= 3; ++i){
            this_thread::sleep_for(chrono::milliseconds(200));
            cout << name << ": " << "Work " << i << " done !!!" << endl;
        }
    }
private:
    string name;
};

...
cout << "Boss: Let's start working.\n\n";
thread herb= thread(Worker("Herb"));
thread andrei= thread(Worker("    Andrei"));
thread scott= thread(Worker("        Scott"));
thread bjarne= thread(Worker("            Bjarne"));
thread andrew= thread(Worker("                Andrew"));
thread david= thread(Worker("                    David"));
    // join all Worker
cout << "\n" << "Boss: Let's go home." << endl;
```

Auswirkung

```
bin : bash - Konsole
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
rainer@icho:~>bossAndWorker

Boss: Let's start working.

    Bjarne: Work 1  Andrei done !!!! Work          David1: Work
Herb: Work 1 done !!!
    Andrew: Work 1 done !!!
1 done !!!
done !!!
    Scott: Work 1 done !!!
    Andrei: Work 2 done !!!
Herb: Work 2 done !!!
    Bjarne: Work 2 done !!!
    Andrew    Scott: Work 2 done !!!
: Work 2 done !!!
    David: Work 2 done !!!
    Andrei: Work 3 done !!!
    Andrew: Work 3 done !!!
    Bjarne: Work 3 done !!!
    Scott: Work 3 done !!!
    David: Work 3 done !!!
Herb: Work 3 done !!!

Boss: Let's go home.

rainer@icho:~>
```

Mutex

```
mutex coutMutex;  
  
struct Worker{  
    Worker(string n):name(n){};  
    void operator() (){  
        for (int i= 1; i <= 3; ++i){  
            sleep_for(milliseconds(200));  
  
            coutMutex.lock();  
            cout << name << ": " << "Work " <<  
            i << " done !!!" << endl;  
            coutMutex.unlock();  
        }  
    }  
private:  
    string name;  
};
```

std::cout ist threadsicher



```
bin : bash - Konsole  
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe  
rainer@icho:~>bossWorkerMutex  
Boss: Let's start working.  
Herb: Work 1 done !!!  
    Bjarne: Work 1 done !!!  
    Scott: Work 1 done !!!  
    Andrei: Work 1 done !!!  
    Andrew: Work 1 done !!!  
    David: Work 1 done !!!  
Herb: Work 2 done !!!  
    Scott: Work 2 done !!!  
    Bjarne: Work 2 done !!!  
    David: Work 2 done !!!  
    Andrew: Work 2 done !!!  
    Andrei: Work 2 done !!!  
    Bjarne: Work 3 done !!!  
Herb: Work 3 done !!!  
    Scott: Work 3 done !!!  
    Andrew: Work 3 done !!!  
    David: Work 3 done !!!  
    Andrei: Work 3 done !!!  
Boss: Let's go home.  
rainer@icho:~>
```

Immer noch ein Problem?

```
void operator() (){
    for (int i= 1; i <= 3; ++i){
        sleep_for(milliseconds(200));
        coutMutex.lock();
        cout << name << ": " << "Work "
        << i << " done !!!" << endl;
        coutMutex.unlock();
    }
}
```

~~Mutex~~ → Lock



```
void operator() (){
    for (int i= 1; i <= 3; ++i)
        sleep_for(milliseconds(200));
    lock_guard<mutex>coutGuard(coutMutex);
    cout << name << ": " << "Work " << i <<
    " done !!!" << endl;
}
```

Problem? und Auswirkung!

```
struct CriticalData{
    mutex mut;
};

void lock(CriticalData& a, CriticalData& b){
    lock_guard<mutex>guard1(a.mut);
    cout << "get the first mutex" << endl;
    this_thread::sleep_for(chrono::milliseconds(1));
    lock_guard<mutex>guard2(b.mut);
    cout << "get the second mutex" << endl;
    // do something with a and b
}

int main(){
    CriticalData c1;
    CriticalData c2;
    thread t1([&]{lock(c1,c2);});
    thread t2([&]{lock(c2,c1);});
    t1.join();
    t2.join();
}
```

A terminal window titled "bin : bash - Konsole" showing the output of the C++ program. The output is "get the first mutexget the first mutex", which is a result of a deadlock. The user has entered "deadlock" and pressed Enter, and then Ctrl-C (^C) to interrupt the program. The terminal window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Lesezeichen", "Einstellungen", and "Hilfe". The status bar at the bottom shows "bin : bash".

```
bin : bash - Konsole
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
rainer@icho:~>deadlock
get the first mutexget the first mutex
^C
rainer@icho:~>
```

unique_lock

```
void deadLock(CriticalData& a, CriticalData& b){  
  
    unique_lock<mutex>guard1(a.mut,defer_lock);  
    cout << "Thread: " << get_id() <<" first mutex"<< endl;  
  
    this_thread::sleep_for(chrono::milliseconds(1));  
  
    unique_lock<mutex>guard2(b.mut,defer_lock);  
    cout <<"      Thread: " << get_id() <<" second mutex"<< endl;  
  
    cout <<"          Thread: " << get_id() <<" get both mutex" << endl;  
  
    lock(guard1,guard2);  
  
}
```



```
bin : bash - Konsole <2>  
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe  
rainer@icho:~>deadlockResolved  
  
Thread: 139991456761600 first mutex  
Thread: 139991448368896 first mutex  
      Thread: 139991456761600 second mutex  
      Thread: 139991456761600 get both mutex  
      Thread: 139991448368896 second mutex  
      Thread: 139991448368896 get both mutex  
  
rainer@icho:~>
```

Problem? und Auswirkung

```
int main(){

    std::thread t([]{
        std::cout << "Still waiting ..." << std::endl;
        std::lock_guard<std::mutex> lockGuard(coutMutex);
        std::cout << std::this_thread::get_id() << std::endl;
    });

    {
        std::lock_guard<std::mutex> lockGuard(coutMutex);
        std::cout << std::this_thread::get_id() << std::endl;
        t.join();
    }
}
```

A screenshot of a terminal window titled "bin : blockJoin - Konsole". The window has a menu bar with "File", "Edit", "View", "Bookmarks", "Settings", and "Help". The terminal output shows the program execution: "rainer@linux: ~>blockJoin", followed by the thread ID "139962613421888", and then "Still waiting ...". The terminal has a yellow background and a black cursor. At the bottom, there is a taskbar with a tab labeled "bin : blockJoin" and a close button.

join frühzeitig

```
int main(){

    std::thread t([]{
        std::cout << "Still waiting ..." << std::endl;
        std::lock_guard<std::mutex> lockGuard(coutMutex);
        std::cout << std::this_thread::get_id() << std::endl;
    });

    {
        t.join();
        std::lock_guard<std::mutex> lockGuard(coutMutex);
        std::cout << std::this_thread::get_id() << std::endl;
    }
}
```

A terminal window titled "bin : bash - Konsole" showing the execution of a program. The output is as follows:

```
File Edit View Bookmarks Settings Help
rainer@linux:~>blockJoin
Still waiting ...
139637574399744
139637591021376
rainer@linux:~>
```

Lösung, aber ...

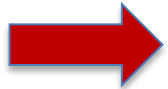
```
mutex myMutex;
```

```
class MySingleton{
public:
    static MySingleton& getInstance(){
        lock_guard<mutex> myLock(myMutex);
        if( !instance ) instance= new MySingleton();
        return *instance;
    }
private:
    MySingleton();
    ~MySingleton();
    MySingleton(const MySingleton&)= delete;
    MySingleton& operator=(const MySingleton&)= delete;
    static MySingleton* instance;
};

MySingleton::MySingleton()= default;
MySingleton::~~MySingleton()= default;
MySingleton* MySingleton::instance= nullptr;
...
MySingleton::getInstance();
```

Double-Checked Locking Pattern

```
class MySingleton{
    static mutex myMutex;
public:
    static MySingleton& getInstance(){
        if ( !instance ){
            lock_guard<mutex> myLock(myMutex);
            if( !instance ) instance= new MySingleton();
            return *instance;
        }
    }
    ...
}
```



`instance= new MySingleton();` ist nicht atomar

1. Allokiere Speicher für MySingleton
2. Erzeuge MySingleton Objekt in dem Speicher
3. Verweise instance auf das MySingleton Objekt

Mögliche Ausführungsreihenfolge 1, 3 und 2

call_once oder Meyers Singleton

call_once und once_flag

```
class MySingleton{
public:
    static MySingleton& getInstance(){
        call_once(initInstanceFlag,
            &MySingleton::initSingleton);
        return *instance;
    }
private:
    ...
    static once_flag initInstanceFlag;
    static void initSingleton(){
        instance= new MySingleton;
    }
};
...
once_flag MySingleton::initInstanceFlag;
```

Meyers Singleton

```
class MySingleton{
public:
    static MySingleton& getInstance(){
        static MySingleton instance;
        return instance;
    }
private:
    ...
```

Atomare Variablen

Thread lokale Daten



- Gehören exklusiv einem Thread
- Jeder Thread hat eine eigene Referenz
- Verhalten sich wie statische Variablen
 - werden bei der ersten Nutzung erzeugt
 - sind an die Lebenszeit des Threads gebunden

Probleme?

```
mutex coutMutex;  
thread_local string s("hello from ");  
  
void addThreadLocal(string const& s2){  
    s+=s2;  
    lock_guard<mutex> guard(coutMutex);  
    cout << s << endl;  
    cout << "&s: " << &s << endl;  
    cout << endl;  
}  
  
int main(){  
    thread t1(addThreadLocal,"t1");  
    thread t2(addThreadLocal,"t2");  
    thread t3(addThreadLocal,"t3");  
    thread t4(addThreadLocal,"t4");  
  
    t1.join(), t2.join(), t3.join(),t4.join();  
}
```

A screenshot of a terminal window titled 'bin : bash - Konsole <2>'. The window shows the command 'rainer@icho:~>threadLocal' and its output. The output consists of four blocks, each starting with 'hello from' followed by a thread ID (t4, t1, t3, t2) and then '&s: ' followed by a memory address. The addresses are 0x7f50d3cf16f8, 0x7f50d54f46f8, 0x7f50d44f26f8, and 0x7f50d4cf36f8 respectively. The prompt 'rainer@icho:~>' is visible at the bottom.

```
bin : bash - Konsole <2>  
Datei Bearbeiten Ansicht Lesezeichen Einstellungen >  
rainer@icho:~>threadLocal  
  
hello from t4  
&s: 0x7f50d3cf16f8  
  
hello from t1  
&s: 0x7f50d54f46f8  
  
hello from t3  
&s: 0x7f50d44f26f8  
  
hello from t2  
&s: 0x7f50d4cf36f8  
rainer@icho:~>
```

Bedingungsvariablen



www.shutterstock.com - 128157377

- Ermöglicht Threads sich mit Nachrichten zu synchronisieren
- Ein Thread erzeugt ein Ergebnis, auf das ein anderer Thread wartet
- Der Producer benachrichtigt einen oder mehrere Consumer

Probleme?



Spurious Wakeup

```
mutex mutex_;
condition_variable condVar;

void waitingForWork() {
    unique_lock<mutex> lck(mutex_);
    condVar.wait(lck);
}

void setDataReady() {
    condVar.notify_one();
}

int main() {
    thread t1(waitingForWork);
    thread t2(setDataReady);
    t1.join();
    t2.join();
}
```

```
bool dataReady= false;

void waitingForWork() {
    unique_lock<mutex> lck(mutex_);
    condVar.wait(lck, []{return dataReady;});
}

void setDataReady() {
    {
        lock_guard<mutex> lck(mutex_);
        dataReady=true;
    }
    condVar.notify_one();
}
```

Problem?

```
mutex mutex_;
condition_variable condVar;

void waitingForWork() {
    unique_lock<mutex> lck(mutex_);
    condVar.wait(lck, []{return dataReady;});
}

void setDataReady() {
    {
        lock_guard<mutex> lck(mutex_);
        dataReady=true;
    }
    condVar.notify_one();
}

...
thread t1(setDataReady);
// short time delay
thread t2(waitingForWork);

t1.join(), t2.join();
}
```



Lost Wakeup

- The `condition_variable` class is a synchronization primitive that can be used to block a thread, or multiple threads **at the same time**, ...

http://en.cppreference.com/w/cpp/thread/condition_variable

Es gibt viel zu tun.

```
bin : bash - Konsole <2>
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
rainer@icho:~>bossWorker
BOSS: PREPARE YOUR WORK.

    David: Work prepared after 587 milliseconds.
    Bjarne: Work prepared after 804 milliseconds.
    Andrei: Work prepared after 899 milliseconds.
    Scott: Work prepared after 1138 milliseconds.
    Andrew: Work prepared after 1717 milliseconds.
    Herb: Work prepared after 1971 milliseconds.

BOSS: START YOUR WORK.

    Andrew: Work done after 286 milliseconds.
    Scott: Work done after 297 milliseconds.
    Bjarne: Work done after 330 milliseconds.
    Andrei: Work done after 346 milliseconds.
    Herb: Work done after 346 milliseconds.
    David: Work done after 380 milliseconds.

BOSS: GO HOME.

rainer@icho:~>
```

Boss

```
preparedCount.store(0);    // atomic

Worker herb("  Herb"), thread herbWork(herb);
...
Worker david("          David"), thread davidWork(david);

cout << "BOSS: PREPARE YOUR WORK.\n " << endl;

unique_lock<mutex> preparedUniqueLock( preparedMutex );
worker2BossCondVariable.wait(preparedUniqueLock, []{ return preparedCount == 6; });

startWork= true; cout << "\nBOSS: START YOUR WORK. \n" << endl;

doneCount.store(0);        // atomic
boss2WorkerCondVariable.notify_all();

unique_lock<mutex> doneUniqueLock( doneMutex );
worker2BossCondVariable.wait(doneUniqueLock, []{ return doneCount == 6; });

goHome= true;  cout << "\nBOSS: GO HOME. \n" << endl;
boss2WorkerCondVariable.notify_all();

// join all Worker
```

Veranstalter



Gold-Partner



Worker

```
struct Worker{
    Worker(string n):name(n){};
    void operator() (){
        int prepareTime= getRandomTime(500,2000);
        this_thread::sleep_for(milliseconds(prepareTime));
        preparedCount++;
        cout << name << ": " << "Work prepared after " << prepareTime << " milliseconds." << endl;
        worker2BossCondVariable.notify_one();
        { // wait for the boss
            unique_lock<mutex> startWorkLock( startWorkMutex );
            boss2WorkerCondVariable.wait( startWorkLock,[] { return startWork;});
        }
        int workTime= getRandomTime(200,400);
        this_thread::sleep_for(milliseconds(workTime));
        doneCount++;
        cout << name << ": " << "Work done after " << workTime << " milliseconds." << endl;
        worker2BossCondVariable.notify_one();
        { // wait for the boss
            unique_lock<mutex> goHomeLock( goHomeMutex );
            boss2WorkerCondVariable.wait( goHomeLock,[] { return goHome;});
        }
    }
}
....
```



Verwende Tasks.

Promise und Futures



- Sind Kanäle zwischen dem Sender und Empfänger
- Der Producer setzt einen Wert in dem Kanal, auf das der Consumer wartet
- Der Sender wird Promise genannt, der Empfänger Future

Promise und Futures

```
int a= 2000;  
int b= 11;
```

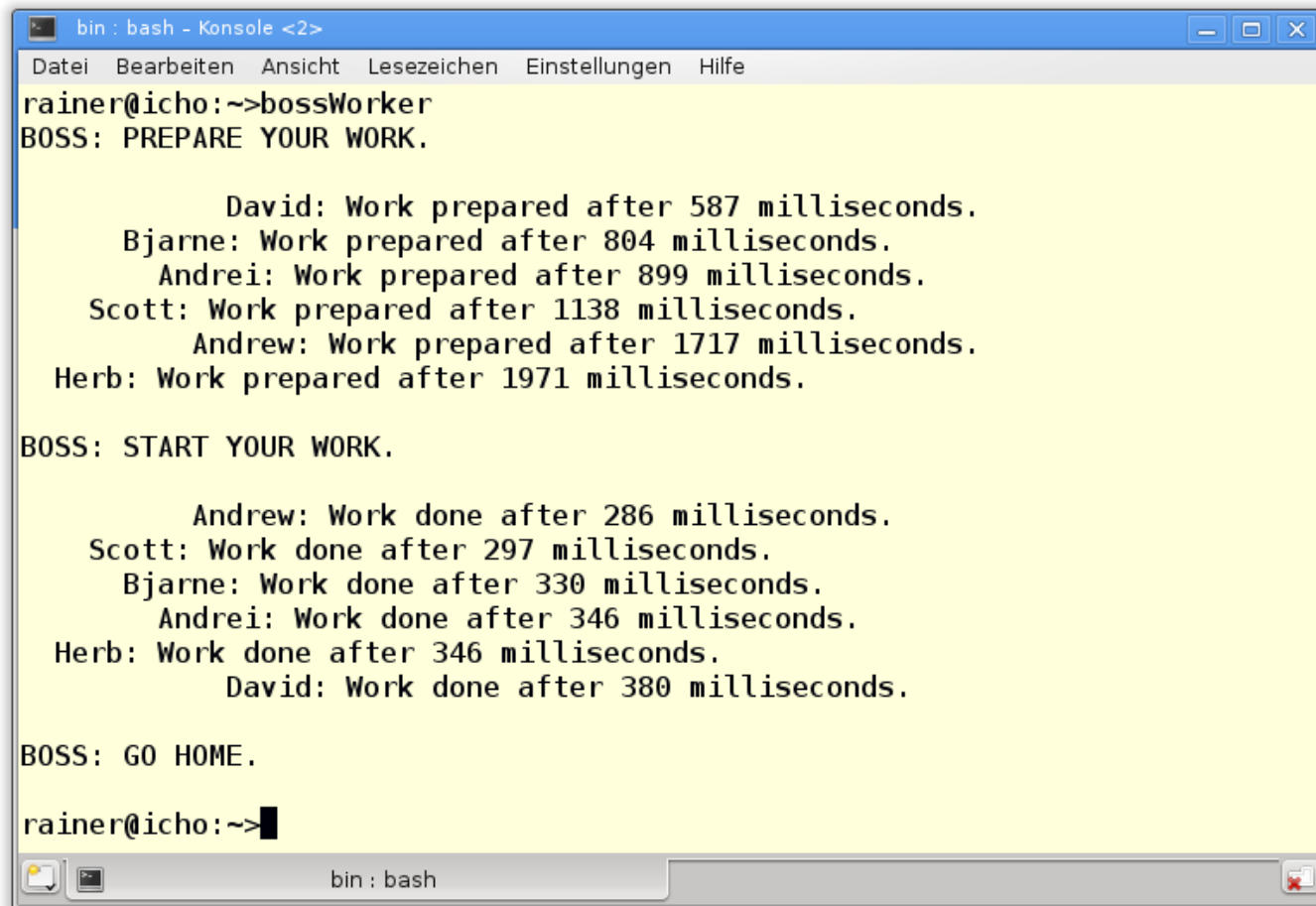
- implizit mit `std::async`

```
future<int> sumFuture= async([=]{ return a+b; });  
sumFuture.get();    // 2011
```

- explizit mit `std::future` und `std::promise`

```
void sum(promise<int>&& intProm, int x, int y){  
    intProm.set_value(x+y);  
}  
promise<int> sumPromise;  
future<int> futRes= sumPromise.get_future();  
thread sumThread(&sum, move(sumPromise), a, b);  
futRes.get();    // 2011
```

Es gibt viel zu tun.



The screenshot shows a terminal window titled "bin : bash - Konsole <2>". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Lesezeichen", "Einstellungen", and "Hilfe". The terminal content is as follows:

```
bin : bash - Konsole <2>
rainer@icho:~>bossWorker
BOSS: PREPARE YOUR WORK.

    David: Work prepared after 587 milliseconds.
    Bjarne: Work prepared after 804 milliseconds.
    Andrei: Work prepared after 899 milliseconds.
    Scott: Work prepared after 1138 milliseconds.
    Andrew: Work prepared after 1717 milliseconds.
    Herb: Work prepared after 1971 milliseconds.

BOSS: START YOUR WORK.

    Andrew: Work done after 286 milliseconds.
    Scott: Work done after 297 milliseconds.
    Bjarne: Work done after 330 milliseconds.
    Andrei: Work done after 346 milliseconds.
    Herb: Work done after 346 milliseconds.
    David: Work done after 380 milliseconds.

BOSS: GO HOME.
rainer@icho:~>
```

The terminal window has a taskbar at the bottom with a "bin : bash" tab and a close button.

Boss

```
promise<void> startWorkProm;  
promise<void> goHomeProm;  
  
shared_future<void> startWorkFut= startWorkProm.get_future();  
shared_future<void> goHomeFut= goHomeProm.get_future();  
  
promise<void> herbPrepared;  
future<void> waitForHerbPrepared= herbPrepared.get_future();  
promise<void> herbDone;  
future<void> waitForHerbDone= herbDone.get_future();  
Worker herb("  Herb");  
thread herbWork(herb,move(herbPrepared),startWorkFut,move(herbDone),goHomeFut);  
...  
  
cout << "BOSS: PREPARE YOUR WORK.\n " << endl;  
waitForHerbPrepared.wait(), waitForScottPrepared.wait(), waitFor ...  
cout << "\nBOSS: START YOUR WORK. \n" << endl;  
startWorkProm.set_value();  
waitForHerbDone.wait(), waitForScottDone.wait(), waitFor ...  
cout << "\nBOSS: GO HOME. \n" << endl;  
goHomeProm.set_value();  
  
// join all Worker
```

Worker

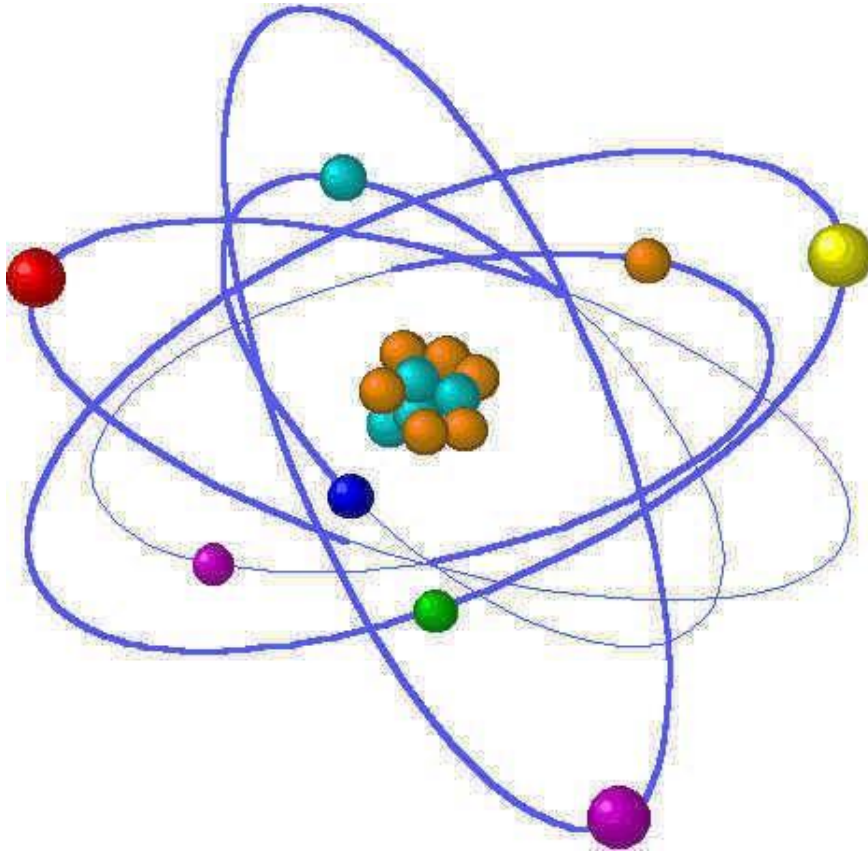
```
struct Worker{
    Worker(string n):name(n){};
    void operator() (promise<void>&& preparedWork, shared_future<void> boss2WorkerStartWork,
                    promise<void>&& doneWork, shared_future<void>boss2WorkerGoHome ){
        int prepareTime= getRandomTime(500,2000);
        this_thread::sleep_for(milliseconds(prepareTime));
        preparedWork.set_value();
        cout << name << ": " << "Work prepared after " << prepareTime << " milliseconds." << endl;

        boss2WorkerStartWork.wait();

        int workTime= getRandomTime(200,400);
        this_thread::sleep_for(milliseconds(workTime));
        doneWork.set_value();
        cout << name << ": " << "Work done after " << workTime << " milliseconds." << endl;

        boss2WorkerGoHome.wait();
    }
private:
    string name;
};
```

Speichermodell



- Das Speichermodell beschäftigt sich mit
 - atomaren Operationen
 - teilweise Ordnung von Operationen
 - der sichtbare Effekt von Operationen

Probleme?

```
int x= 0;  
int y= 0;
```

```
void writing() {  
    x= 2000;  
    y= 11;  
}
```

```
void reading() {  
    cout << "y: " << y << " ";  
    cout << "x: " << x << endl;  
}
```

```
int main() {  
    thread thread1(writing);  
    thread thread2(reading);  
    thread1.join();  
    thread2.join();  
};
```



y	x	Yes
0	0	☐
11	0	☐
0	2000	☐
11	2000	☐

Mutex

```
int x= 0;  
int y= 0;  
mutex mut;
```

```
void writing(){  
    lock_guard<mutex> guard(mut);  
    x= 2000;  
    y= 11;  
}
```

```
void reading(){  
    lock_guard<mutex> guard(mut)  
    cout << "y: " << y << " ";  
    cout << "x: " << x << endl;  
}  
...
```

```
thread thread1(writing);  
thread thread2(reading);
```



y	x	Yes
0	0	☐
11	0	☐
0	2000	☐
11	2000	☐

Probleme?

```
volatile x= 0;  
volatile y= 0;
```

```
void writing(){  
    x= 2000;  
    y= 11;  
}
```

```
void reading(){  
    cout << y << " ";  
    cout << x << endl;  
}
```

...

```
thread thread1(writing);  
thread thread2(reading);
```



y	x	Yes
0	0	☐
11	0	☐
0	2000	☐
11	2000	☐

Java versus C++

Java volatile == C++ atomic

- `std::atomic`
 - Schutz der Daten von gemeinsamen Zugriff mehrerer Threads
- `volatile`
 - Zugriff auf speziellen Speicher, auf dem Lesen und Schreiben nicht optimiert werden darf

Atomare Variablen

```
atomic<int> x= 0;  
atomic<int> y= 0;
```

```
void writing(){  
    x.store(2000);  
    y.store(11);  
}
```

```
void reading(){  
    cout << y.load() << " ";  
    cout << x.load() << endl;  
}
```

...

```
thread thread1(writing);  
thread thread2(reading);
```



y	x	Yes
0	0	☐
11	0	☐
0	2000	☐
11	2000	☐

Acquire-Release Semantik

```
atomic<int> x= 0;  
atomic<int> y= 0;
```

```
void writing() {  
    x.store(2000, memory_order_relaxed);  
    y.store(11, memory_order_release);  
}
```

```
void reading() {  
    cout << y.load(memory_order_acquire) << " ";  
    cout << x.load(memory_order_relaxed) << endl;  
}
```

...

```
thread thread1(writing);  
thread thread2(reading);
```



y	x	Yes
0	0	☐
11	0	☐
0	2000	☐
11	2000	☐

Nicht atomare Variablen

```
int x= 0;  
atomic<int> y= 0;
```

```
void writing() {  
    x= 2000;  
    y.store(11, memory_order_release);  
}
```

```
void reading() {  
    cout << y.load(memory_order_acquire) << " ";  
    cout << x << endl;  
}
```

...

```
thread thread1(writing);  
thread thread2(reading);
```



y	x	Yes
0	0	☐
11	0	☐
0	2000	☐
11	2000	☐

Relaxed Semantik

```
atomic<int> x= 0;  
atomic<int> y= 0;
```

```
void writing(){  
    x.store(2000,memory_order_relaxed);  
    y.store(11, memory_order_relaxed);  
}
```

```
void reading(){  
    cout << y.load(memory_order_relaxed) << " ";  
    cout << x.load(memory_order_relaxed) << endl;  
}
```

...

```
thread thread1(writing);  
thread thread2(reading);
```

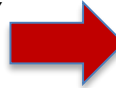


y	x	Yes
0	0	☐
11	0	☐
0	2000	☐
11	2000	☐

Singleton mit atomaren Variablen

Sequentially Consistent

Acquire-Release

```
class MySingleton{
    static mutex myMutex;
    static atomic<MySingleton*> instance;
public:
    static MySingleton* getInstance(){
        MySingleton* sin= instance.load();  instance.load(memory_order_acquire);
        if ( !sin ){
            lock_guard<mutex> myLock(myMutex);
            if( !sin ){
                sin= new MySingleton();
                instance.store(sin);  instance.store(sin,memory_order_release);
            }
        }
        return sin;
    }
    ...
}
```

Singleton mit Fences

Acquire-Release mit Fences

```
class MySingleton{
    static mutex myMutex;
    static std::atomic<MySingleton*> instance;
public:
    static MySingleton* getInstance(){
        MySingleton* sin= instance.load(std::memory_order_relaxed);
        std::atomic_thread_fence(std::memory_order_acquire);
        if ( !sin ){
            lock_guard<mutex> myLock(myMutex);
            if( !sin ){
                sin= new MySingleton();
                std::atomic_thread_fence(std::memory_order_release);
                instance.store(sin,std::memory_order_relaxed);
            }
        }
        return sin;
    }
    ...
}
```

Die Lösung: CppMem

CppMem: Interactive C/C++ memory model

Model

☐ standard ☐ preferred ☒ release_acquire ☐ tot ☐ relaxed_only

Program

examples/MP_message_passing ▾ MP+na_rel+acq_na.c ▾

☒ C ☐ Execution

```
// MP+na_rel+acq_na
// Message Passing, of data held in non-atomic x,
// with release/acquire synchronisation on y.
// Question: is the read of x required to see the new data value 1
// rather than the initial state value 0?
int main() {
  int x=0; atomic_int y=0;
  { { { x=2000;
        y.store(11,memory_order_release); }
    ||| { r1=y.load(memory_order_acquire);
          r2=x; } } }
  return 0;
}
```

run reset help 8 executions; 2 consistent, only 1 race free

Computed executions

Display Relations

- ☒ sb ☐ asw ☐ dd ☐ cd
☒ rf ☒ mo ☒ sc ☒ lo
☐ hb ☐ vse ☐ ithb ☒ sw ☐ rs ☐ hrs ☒ dob ☐ cad
☒ unsequenced_races ☒ data_races

Display Layout

- ☐ dot ☐ neato_par ☒ neato_par_init ☐ neato_downwards
☐ tex
edit display options

Execution candidate no. 1 of 8

previous consistent previous candidate next candidate next consistent 1 goto

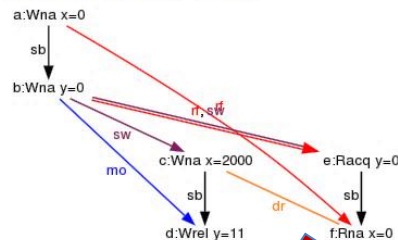
Model Predicates

consistent_race_free_execution = false

☒	consistent_execution	= true
☒	assumptions	= true
☒	well_formed_threads	= true
☒	well_formed_rf	= true
☒	locks_only_consistent_locks	= true
☒	locks_only_consistent_lo	= true
☒	consistent_mo	= true
☒	sc_accesses_consistent_sc	= true
☒	sc_fenced_sc_fences_heeded	= true
☒	consistent_hb	= true
☒	consistent_rf	= true
☒	det_read	= true
☒	consistent_non_atomic_rf	= true
☒	consistent_atomic_rf	= true
☒	coherent_memory_use	= true
☒	rmw_atomicity	= true
☒	sc_accesses_sc_reads_restricted	= true
☒	unsequenced_races	are absent
☒	data_races	are present
☒	indeterminate_reads	are absent
☒	locks_only_bad_mutexes	are absent



[CppMem](#)



Files: out.exc, out.dot, out.dsp, out.t

Vielen Dank!

Rainer Grimm

Veranstalter



Gold-Partner

