**2024 01 10**

#include <shared\_mutex>

**using** **namespace** std**::**literals**;**

**int** cnt **=** 0**;**

std**::**shared\_mutex mtx**;**

**void** writer**()**

**{**

**for** **(int** i **=** 0**;** i **<** 10**;** **++**i**)** **{**

**{**

std**::**scoped\_lock lock**(**mtx**);**

**++**cnt**;**

**}**

std**::**this\_thread**::**sleep\_for**(**100ms**);**

**}**

**}**

**void** reader**()**

**{**

**for** **(int** i **=** 0**;** i **<** 100**;** **++**i**)** **{**

**int** c**;**

**{**

***// yazma amaçlı thread kilidi edinemez ama oku amaçlı edinebilir***

std**::**shared\_lock lock**(**mtx**);**

c **=** cnt**;**

**}**

std**::**osyncstream**{** std**::**cout **}** **<<** std**::**this\_thread**::**get\_id**()** **<<** ' ' **<<** c **<<** '\n'**;**

std**::**this\_thread**::**sleep\_for**(**10ms**);**

**}**

**}**

**int** main**()**

**{**

std**::**vector**<**std**::**jthread**>** tvec**;**

tvec**.**reserve**(**16**);**

tvec**.**emplace\_back**(**writer**);**

**for** **(int** i **=** 0**;** i **<** 16**;** **++**i**)**

tvec**.**emplace\_back**(**reader**);**

**}**

***// std::unique\_lock***

std**::**mutex mtx**;**

**int** main**()**

**{**

unique\_lock**<**mutex**>** ulock**;**

***// diğer lock sınıflarından farklı olarak aşağıdakilere sahip:***

ulock**.**lock**();**

ulock**.**unlock**();**

ulock**.**try\_lock**();**

***// unique\_lock move only type***

**}**

**using** **namespace** std**;**

std**::**mutex mtx**;**

**void** foo**()**

**{**

unique\_lock lock**{** mtx **};** ***// kilidi ediniyoruz lock() çağrılır***

**}**

**void** bar**()**

**{**

***// lock() çağrılmaz***

unique\_lock lock**{** mtx**,** adopt\_lock**};** ***// kilidi edinmiş durumda alıyoruz***

**}**

**void** bar**()**

**{**

***// lock() çağrılmaz***

unique\_lock lock**{** mtx**,** defer\_lock**};** ***// kilidi kitlemiyor.***

***// burada kilitlemek gerekiyor.***

**}**

**void** bam**()**

**{**

***// kilidi edinmeye çalışıyor edinemezse bloke edilmiyor***

unique\_lock lock**{** mtx**,** try\_to\_lock**};**

**if** **(**lock**.**owns\_lock**())**

**{**

***// kilidi edinmişse bu bölgeye girer.***

lock**.**unlock**();**

**}**

**}**

**int** cnt**{};**

std**::**mutex mtx**;**

**void** func**()**

**{**

**for** **(int** i **=** 0**;** i **<** 1'000'000**;** **++**i**)**

**{**

std**::**unique\_lock ulock**{** mtx**,** std**::**defer\_lock **};**

ulock**.**lock**();**

**++**cnt**;**

ulock**.**unlock**();**

ulock**.**lock**();**

**++**cnt**;**

ulock**.**unlock**();**

**}**

**}**

**int** main**()**

**{**

**{**

std**::**jthread t1**{** func **};**

std**::**jthread t2**{** func **};**

std**::**jthread t3**{** func **};**

std**::**jthread t4**{** func **};**

std**::**jthread t5**{** func **};**

**}**

std**::**cout **<<** cnt **<<** "\n"**;**

**}**

***// std::once\_flag and std::call\_once***

**using** **namespace** std**;**

unique\_ptr**<**string**>** uptr**;**

once\_flag flag**;**

**void** initialize**()**

**{**

osyncstream**{** cout **}** **<<** "initialize " **<<** **<**this\_thread**::**get\_id**()** **<<** "\n"**;**

uptr **=** make\_unique**<**string**>(**"emre bahtiyar"**);**

**}**

**const** string**&** get\_value**()**

**{**

***// herhangi bir thread sadece bir kez initialize fonksiyonu çağıracak***

call\_once**(**flag**,** initialize**);**

**return** **\***uptr**;**

**}**

**void** workload**()**

**{**

**const** std**::**string**&** rs **=** get\_value**();**

osyncstream**{** cout **}** **<<** **&**rs **<<** "\n"**;**

**}**

**int** main**()**

**{**

vector**<**thread**>** tvec**;**

tvec**.**reverse**(**20**);**

**for** **(int** i **=** 0**;** i **<** 16**;** **++**i**)**

**{**

tvec**.**emplace\_back**(**workload**)**

**}**

**}**

***// thread-safe singleton***

**class** Singleton **{**

**public:**

Singleton**(const** Singleton**&)** **=** **delete;**

Singleton**&** **operator=(const** Singleton**&)** **=** **delete;**

**static** Singleton**\*** get\_instance**()**

**{**

call\_once**(**m\_init\_flag**,** Singleton**::**init**);**

**return** m\_instance**;**

**}**

**static** **void** init**()**

**{**

m\_instance **=** **new** Singleton**();**

**}**

**private:**

**static** std**::**once\_flag m\_init\_flag**;**

**static** Singleton**\*** m\_instance**;**

Singleton**()** **=** **default;**

**};**

Singleton**\*** Singleton**::**m\_instance**{};**

std**::**once\_flag Singleton**::**m\_init\_flag**;**

**void** func**()**

**{**

std**::**osyncstream**{** std**::**cout **}** **<<** Singleton**::**get\_instance**()** **<<** '\n'**;**

**}**

**int** main**()**

**{**

std**::**vector**<**std**::**thread**>** tvec**;**

**for** **(int** i **=** 0**;** i **<** 100**;** **++**i**)** **{**

tvec**.**emplace\_back**(**func**);**

**}**

**for** **(auto&** th **:** tvec**)**

th**.**join**();**

**}**

***// thread-safe singleton***

**class** Singleton **{**

**public:**

**static** Singleton**&** get\_instance**()**

**{**

**static** Singleton s**;**

**return** s**;**

**}**

**}**

***// call\_once alternatif***

**using** **namespace** std**;**

**void** func**()**

**{**

osyncstream**{**cout**}** **<<** "func cagrildi " **<<** this\_thread**::**get\_id**()** **<<** "\n"**;**

**}**

**void** foo**()**

**{**

std**::**this\_thread**::**sleep\_for**(**50ms**);**

**static** **auto** f **=** **[]** **{**func**();** **return** 0**;}();**

**}**

**int** main**()**

**{**

vector**<**jthread**>** tvec**;**

**for(int** i **=** 0**;** i**<** 10**;** **++**i**)**

tvec**.**emplace\_back**(**foo**);**

**}**

**std::future and std::promise**

#include <future>

**int** main**()**

**{**

**using** **namespace** std**;**

std**::**promise**<int>** prom**;**

future**<int>** ft **=** prom**.**get\_future**();**

prom**.**set\_value**(**12**);**

**int** val **=** ft**.**get**();** ***// val 12***

**}**

**void** produce**(**std**::**promise**<double>** prm**,** **double** val**)**

**{**

prm**.**set\_value**(**dval **\*** dval**);**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

promise**<double>** prom**;**

**auto** ft **=** prom**.**get\_future**();**

thread t**{**produce**,** move**(**prom**),** 4.543**};**

**auto** val **=** ft**.**get**();**

cout **<<** "value = " **<<** val **<<** "\n"**;**

t**.**join**();**

**}**

std**::**string foo**(**std**::**string str**)**

**{**

**auto** temp **=** str**;**

reverse**(**str**.**begin**(),** std**.**end**());**

**return** temp **+** str**;**

**}**

**void** bar**(**std**::**promise**<**std**::**string**>&&** prom**,** std**::**string str**)**

**{**

prom**.**set\_value**(**foo**(**str**));**

**}**

**using** **namespace** std**;**

**int** main**()**

**{**

promise**<**string**>** prom**;**

future**<**string**>** ft **=** prom**.**get\_future**();**

thread t**{**bar**,** move**(**prom**),** "tamer" **};**

cout **<<** ft**.**get**()** **<<** "\n"**;**

t**.**join**();**

**}**

**struct** Div

**{**

**void** **operator()(**std**::**promise**<int>&&** prom**,** **int** a**,** **int** b**)**

**{**

**if** **(**b **==** 0**)**

**{**

**auto** str **=** "divide by zero error " **+** std**::**to\_string**(**a**)** **+** "\\" **+** std**::**to\_string**(**b**);**

prom**.**set\_exception**(**std**::**make\_exception\_ptr**(**std**::**runtime\_error**(**str**));**

**}**

**else**

**{**

prom**.**set\_value**(**a **/** b**);**

**}**

**}**

**};**

**using** **namespace** std**;**

**int** main**()**

**{**

promise**<int>** prom**;**

**auto** ft **=** prom**.**get\_future**();**

thread th**{** Div**{},** move**(**prom**),** 12**,** 3**};**

**try**

**{**

cout **<<** ft**.**get**()** **<<** "\n"**;**

**}**

**catch** **(const** std**::**exception**&** ex**)**

**{**

std**::**cout **<<** "exception caugth : " **<<** ex**.**what**()** **<<** "\n"**;**

**}**

th**.**join**();**

**}**

**Std::async**

***// std::async***

**int** foo**(int** x**,** **int** y**)**

**{**

**return** x **\*** y **+** 5**;**

**}**

**void** bar**(int** x**,** **int** y**)**

**{**

**return** x **+** y **+** 2**;**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

**auto** ft **=** async**(**foo**,** 10**,** 30**);** ***// std::future<int> döner***

**auto** ft1 **=** async**(**bar**,** 10**,** 30**);** ***// std::future<int> döner***

**auto** val **=** ft**.**get**()** **+** ft1**.**get**();**

**}**

***// std::packaged\_task***

**int** foo**(int,** **int);**

**int** main**()**

**{**

**using** **namespace** std**;**

packaged\_task mytask**(**foo**,** 2**,** 5**);**

packaged\_task **<int(int,** **int)>** task**{** foo **};**

thread th**{** task**,** 3**,** 6**};**

**}**