Concurrent programming and memory consistency laboratory

ARCOS Group

Computer Architecture

Bachelor In Informatics Engineering

Universidad Carlos III de Madrid

Objective

Increase the understanding:

- **Lock free programming**
  - **Impact over the performance of an application**

usage of certain types of atomic data will be evaluated versus the usage of techniques based on locks

.

ARCOS @ UC3M

Description of the laboratory

Different alternatives will be evaluated in order to implement **Circular bounded buffer.**

Two techniques:

- lock based programming

- free lock programming

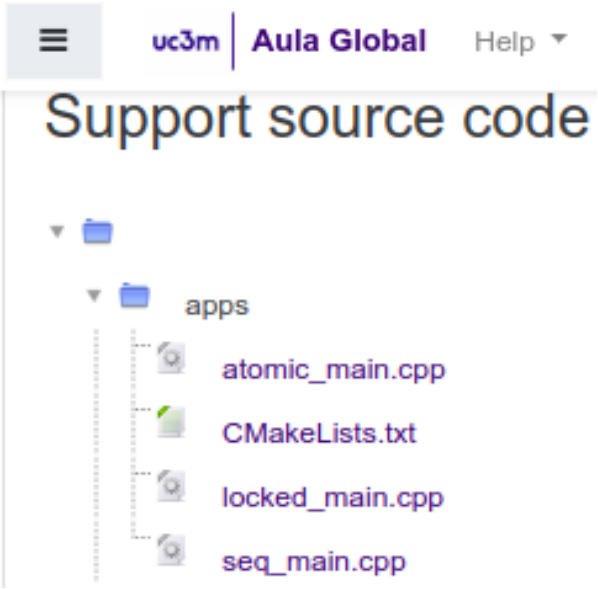
Standard:

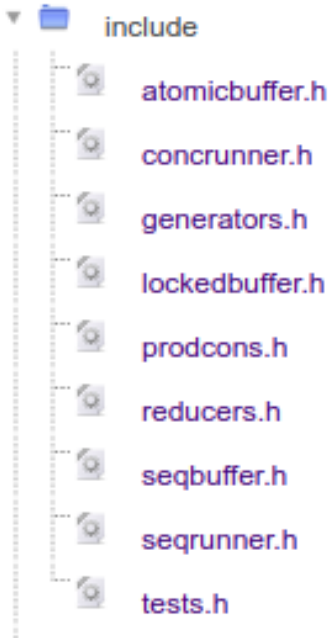
ISO/IEC 14882:2017 (C++17)

ARCOS @ UC3M

Materials supplied

Circular bounded buffer.

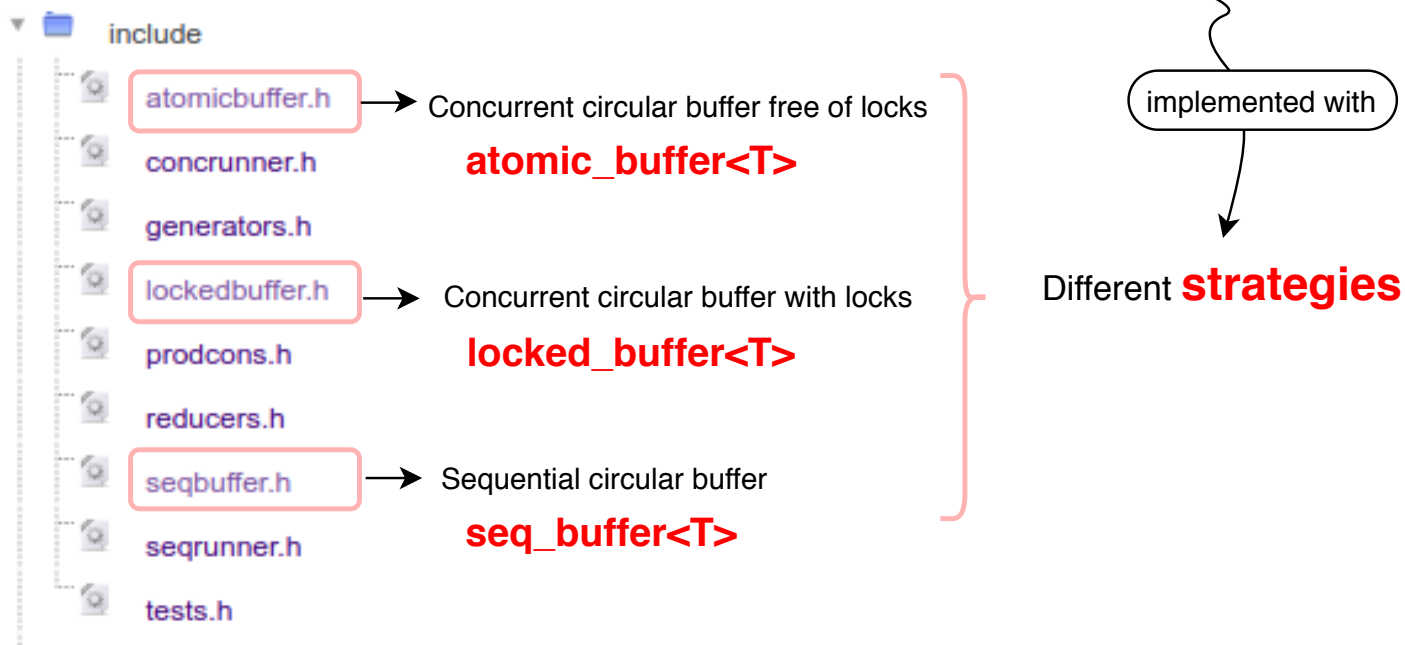




ARCOS @ UC3M

Materials supplied

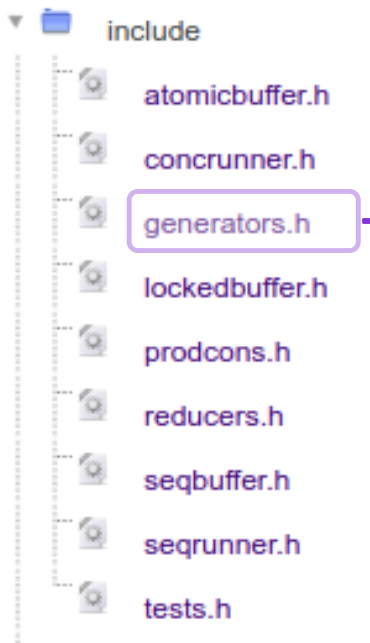
Circular bounded buffer



ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generators



1. **random_numeric_generator<T>**

   Random number sequence.

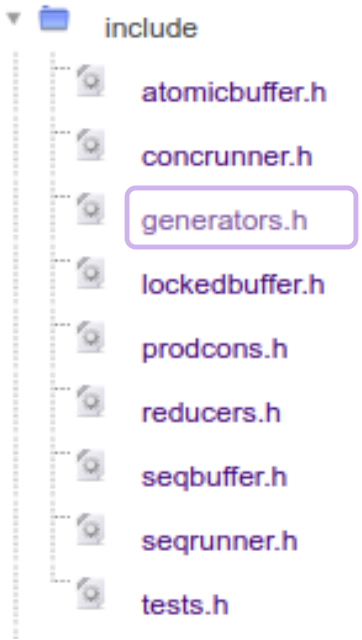
2. **file_generator<T>**

   Sequence of values read from a text file.

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generators



① **random_numeric_generator<T>** ② **file_generator<T>**

Behavior: function objects (functors)

Constructor:

```
① random_numeric_generator<long> rand_gen{10}; // It generates up to 10 long values
② file_generator<string> word_gen{"file.txt"}; // It generates words read from the file "file.txt"
② file_generator<long> num_gen{"num.txt"}; // It generates numbers read from the file "num.txt"
```

Invoked as a function:

```
① auto numval = rgen();
② auto word = fgen();
```

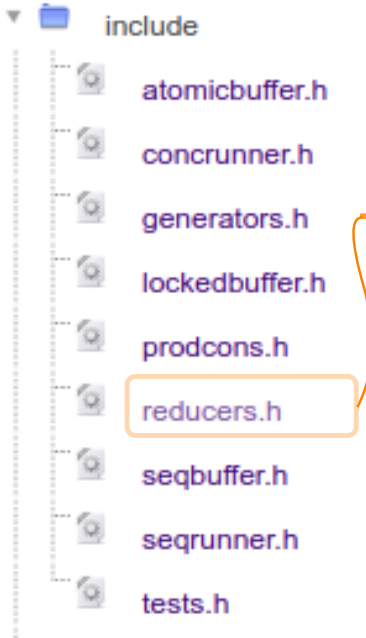
Returns: optional value

```
① optional<long> numval = rgen();
  long val = -1;
  if (numval) { val = *numval; }

② optional<string> word = fgen();
  if (word) {
    cout << *word << "\n";
  }
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Reducers



1. **numeric_reducer<T>**

   Obtain the minimum and maximum.

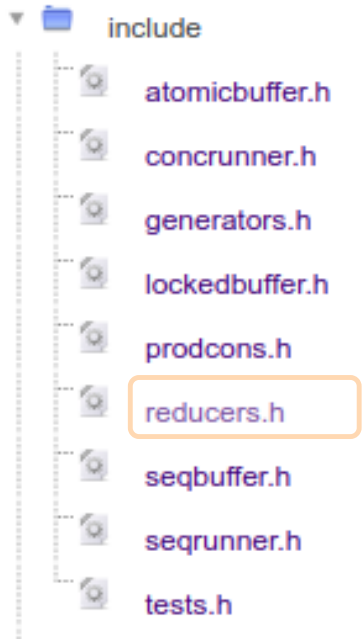
2. **word_counter**

   Counts the frequency of the words.

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Reducers



① numeric_reducer<T> ② word_counter

Constructor:

```
① numeric_reducer<long> num_red; // Numerical reducer for long
② word_counter freq_red; // Word frequency reducer
```

The operator += redefined

```
① num_red += 10L; // Aggregates the value 10L to num_red
  num_red += 30L // Aggregates the value 30L to num_red;
  num_red += 20L // Aggregates the value 20L to num_red;
```

```
② freq_red += "Hola"s;
  freq_red += "C++"s:
  freq_red += "Hola"s;
```

Maximum and Minimum

```
① long a = num_red.max();
  long b = num_red.min();
```

The most frequent word and the number of ocurrences

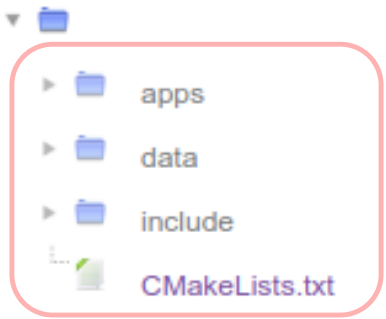
```
② auto r = freq_red.most_frequent();
  cout << "palabra: " << r.first << "\n";
  cout << "frecuencia: " << r.second << "\n";
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer

Support source code



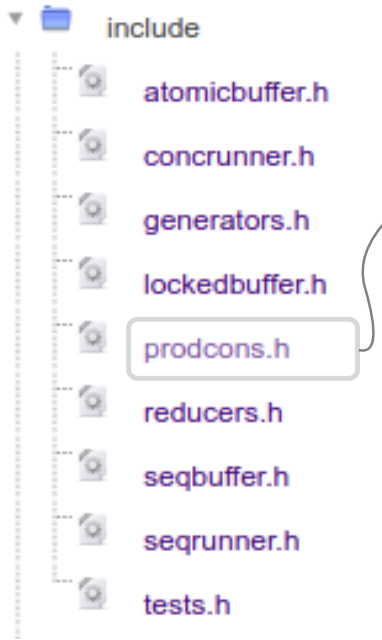
IMPORTANT

It is **recommended** that the student **review** the implementation of these classes, using if necessary the **documentation of the standard library** (for example in http://en.cppreference.com/w/) or recommended textbooks of the subject (C++ Concurrency in Action. Practical multithreading)

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generic Producer



1. class **producer**
   configured with a value **generator** type and a **buffer** type

Generators:

1. random_numeric_generator<T>
2. file_generator<T>

Buffer type (strategies):

1. seq_buffer
2. locked_buffer
3. atomic_buffer<T>

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generic Producer

① class **producer**

configured with a value **generator** type and a **buffer** type

Generators:

① random_numeric_generator<T>

② file_generator<T>

Buffer type (strategies):

① seq_buffer

② locked_buffer

③ atomic_buffer<T>

Example Producer with ① and ② :

```
① using gen_type = numeric_generator<long>;
② using buf_type = locked_buffer<long>;
① gen_type gen{1000}; // Number generator
② buf_type buf{10}; // Locked buffer of size 10

① producer<gen_type,buf_type> prod{gen,buf}; // Productor that uses gen
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generic Producer

Invoke a producer:

ⓐ not pass any argument

```
producer<gen_type,buf_type> p{gen,buf};
ⓐ p(); // Generates values until the end of the sequence

producer<gen_type,buf_type> q{gen,buf};
thread t{q}; // It created a thread that generates values in buf
t.join();
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generic Producer

Invoke a producer:

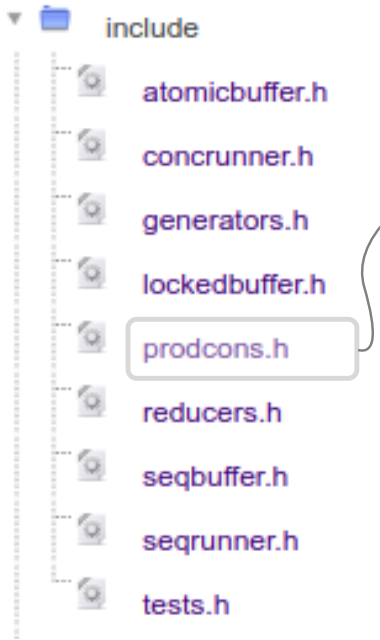
(b) passing as argument a predicate

```
(2) file_generator<string> gen{"texto.txt"};
(1) seq_buffer<string> buf{32};

producer<file_generator<string>,seq_buffer<string>> prod{gen,buf};
bool finisehd = false;
while (!finished) {
(b) prod([&] { return !buf.full(); });
    bool finished = consume(buf);
}
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generic Consumer



class **consumer**

② configured with a **reducer** type and a **buffer** type

Reducers:

1. numeric_reducer<T>
2. word_counter

Buffer type (strategies):

1. seq_buffer
2. locked_buffer
3. atomic_buffer<T>

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generic Consumer

② class consumer

configured with a **reducer** type and a **buffer** type

Reducers:

① numeric_reducer<T>

② word_counter

Buffer type (strategies):

① seq_buffer

② locked_buffer

③ atomic_buffer<T>

Example Consumer with ② and ③ :

```
② word_counter wc;
③ atomic_buffer<string> buf{16};

② consumer<word_counter,atomic_buffer<string>> cons{wc,buf};
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generic Consumer

Invoke a consumer:

(a) not pass any argument

(2) `consumer<word_counter,atomic_buffer<string>> cons{wc,buf};`
(a) `cons(); // It consumes values until the end of the sequence`
`consumer <word_counter, atomic_buffer<string>> q{wc,buf};`
`thread t{q}; // It creates a thread to consume values of buf`

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Generic Consumer

Invoke a consumer:

(b) passing as argument a predicate

Generators:

(1) random_numeric_generator<T>

(2) file_generator<T>

Reducers:

(1) numeric_reducer<T>

(2) word_counter

Buffer type (strategies):

(1) seq_buffer

(2) locked_buffer

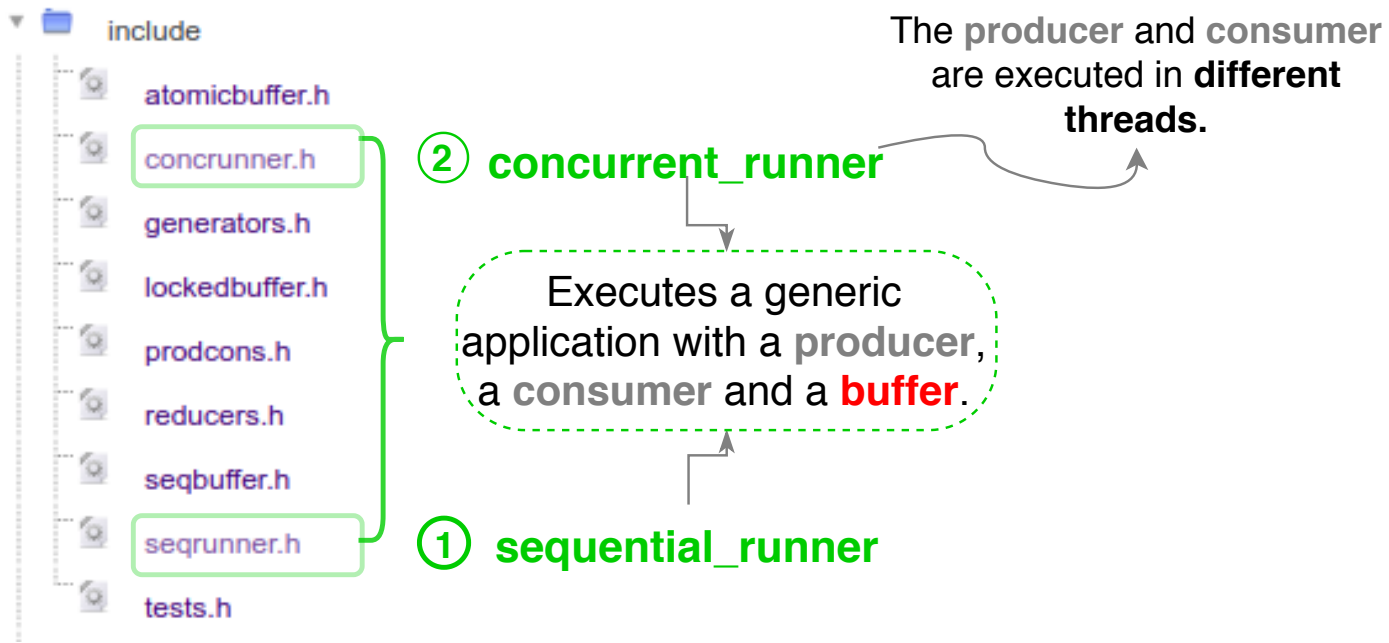
(3) atomic_buffer<T>

```
(2) using gen_type = file_generator<string>;
(2) gen_type gen{"text.txt"};
(2) using red_type = word_counter;
(2) red_type wc;

(2) using buf_type = locked_buffer<string>;
(2) buf_type buf{40};

(1) producer<gen_type,buf_type> prod{gen,buf};
(2) consumer<cons_type, buf_type> cons{red,buf};
    for (;;) {
      prod([&]{ return !buf.full();} );
(2)(b) bool finished = cons([&]{return !buf.empty();} )
      if (finished) break;
    }
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Execution alternatives



ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Execution alternatives

Example (2)
concurrent_runner

Generators:

(1) random_numeric_generator<T>
(2) file_generator<T>

Reducers:

(1) numeric_reducer<T>
(2) word_counter

Buffer type (strategies):

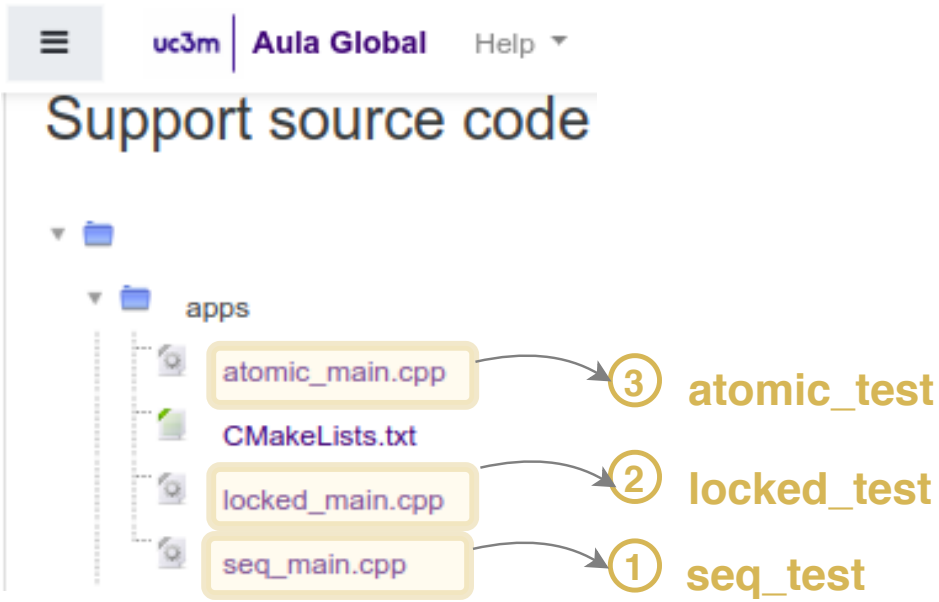
(1) seq_buffer
(2) locked_buffer
(3) atomic_buffer<T>

```
void most_frequent(const std::string & filename) {
(2) locked_buffer<string> buf{20};
(2) file_generator<string> gen{filename};
(2) word_counter wc;

(2) concurrent_runner runner;
(2) runner(gen,wc,buf);

    auto res = reducer.most_frequent();
    cout << "Palabra: " << res.first << "\n";
    cout << "Apariciones: " << res.second << "\n";
}
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Evaluation programs

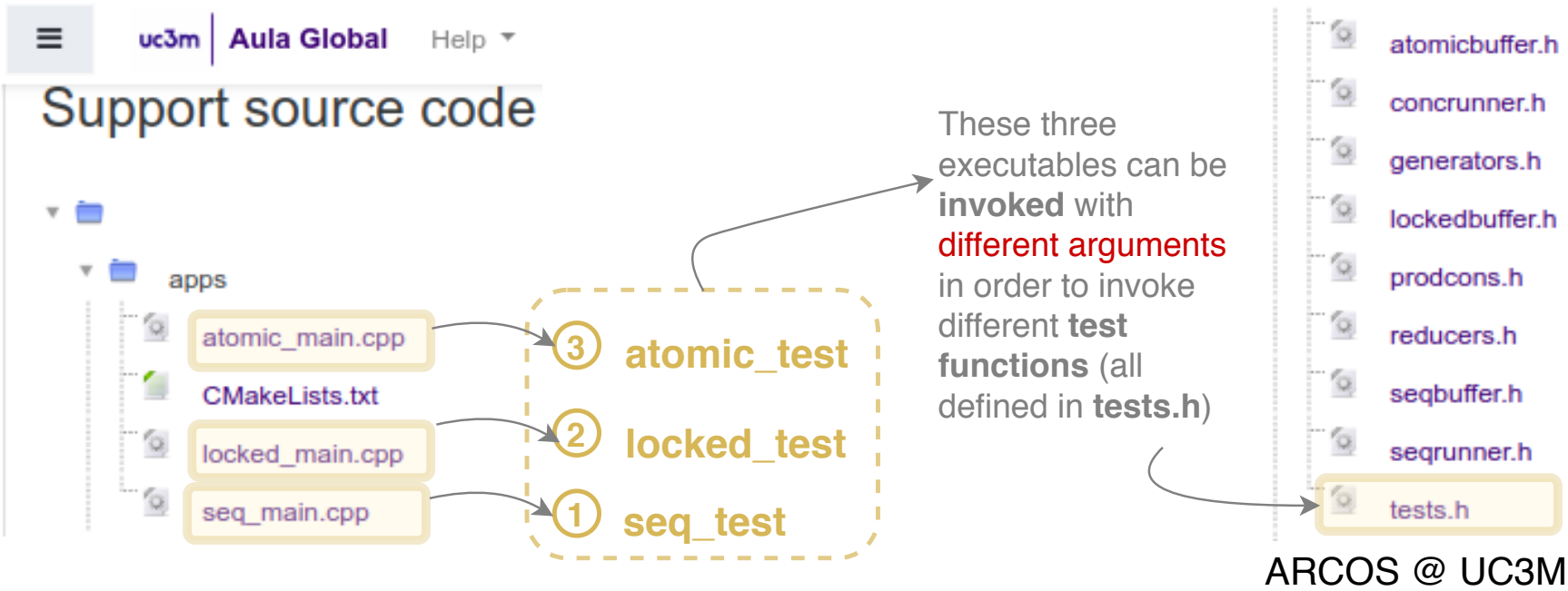


1. seq_test
   - 1 sequential_runner
   - 1 seq_buffer
2. locked_test
   - 2 concurrent_runner
   - 2 locked_buffer
3. atomic_test
   - 2 concurrent_runner
   - 3 atomic_buffer<T>

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Evaluation programs



ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Evaluation programs

Note: The rest of this section **prog** refers indistinctly to any of the three programs (**seq_test, locked_test**, or **atomic_test**).

* **Maximum and minimum of random numbers.**

  Ex:

```
prog random 10 1000
```

* **Maximum and minimum of numbers in a file.**

  Ex:

```
prog file 10 datos.txt
```

* **Words frequency.**

  Ex:

```
prog count 10 ../data/quijote.txt
```

ARCOS @ UC3M

Materials supplied

Circular bounded buffer - Performance Evaluation

* Measure of the **time** using the standard library of C++ (namespace **chrono**).
    * Accessing the Linux kernel module **perf.**

IMPORTANT: Do not forget to activate the optimizations of the compiler before executing the evaluation (mode **Release** of **CMake**).

ARCOS @ UC3M

3. Tasks

3.1 Source code study

3.1.1 Sequential buffer - Answer 9 questions

3.1.2 Locked buffer - Answer 13 questions

3.1.3 Lock free buffer - Answer 12 questions

3. Tasks

3.2 Performance Evaluation

3.2.1 Evaluation with **random** (1000 and 10^6 values / **buffer size** = {2,10,100,1000})

3.2.2 Evaluation with **count** (**quijote.txt** and **king-lear.txt** | **buffer size** = {2,10,100,1000})

3.2.3 Compilation with **CMake**

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
```

Submission

Submission

Deadline → **December 15th**

Quiz:

- Available through Aula Global.
- Individual
- Maximum of 20 minutes to complete it
- One attempt
- Maximum of 10 questions for one student

ARCOS @ UC3M