Synchronizacja wątków za pomocą barier <barrier> i semaforów <semaphore>

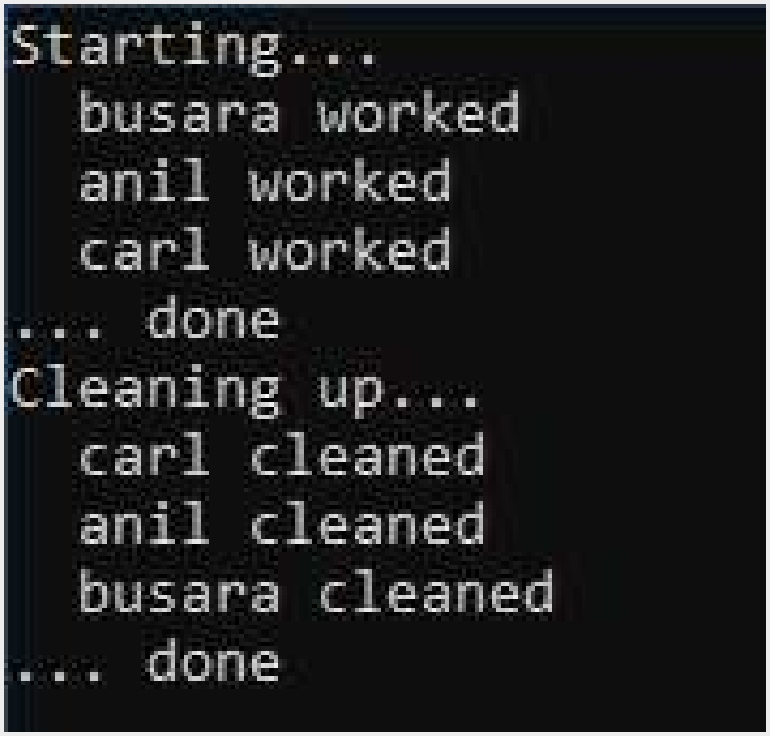
<barrier>

- barrier(std::ptrdiff_t expected, CompletionFunction f) – gdzie expected to oczekiwana liczba wątków, po których zostanie wykonane f,
- arrive(std::ptrdiff_t n) – dekrementuje oczekiwany licznik o n,
- wait() – zamraża do chwili zakończenia podanej wcześniej f,
- arrive_and_wait() – tożsame z wait(arrive()) – dekrementuje licznik o 1, a następnie wstrzymuje dalsze wykonywanie do chwili osiągnięcia oczekiwanej liczby wątków,
- arrive_and_drop() – dekrementuje początkową oczekiwaną liczbę dla wszystkich faz o jeden, a następnie dekrementuje oczekiwaną liczbę dla bieżącej fazy.

Przykład

```
const auto workers = { "anil", "busara", "carl" };

auto on_completion = []() noexcept {
    // locking not needed here
    static auto phase = "... done\n" "Cleaning up...\n";
    std::cout << phase;
    phase = "... done\n";
};
std::barrier sync_point(std::ssize(workers), on_completion);

auto work = [&](std::string name) {
    std::string product = "  " + name + " worked\n";
    std::cout << product;
    sync_point.arrive_and_wait();

    product = "  " + name + " cleaned\n";
    std::cout << product;
    sync_point.arrive_and_wait();
};

std::cout << "Starting...\n";
std::vector<std::thread> threads;
for (auto const& worker : workers) {
    threads.emplace_back(work, worker);
}
for (auto& thread : threads) {
    thread.join();
}
```

Możliwy wynik



<semaphore>

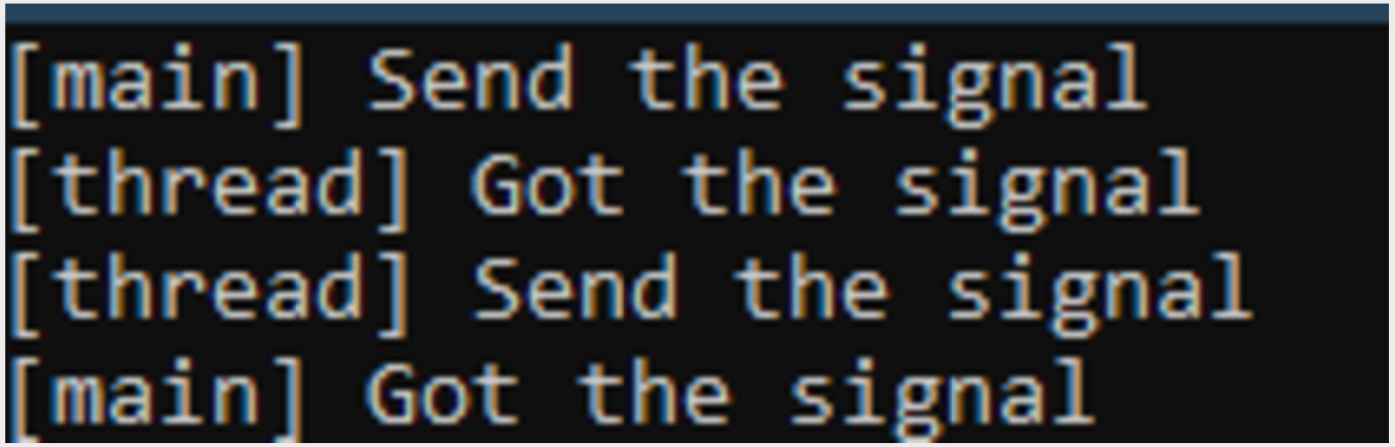
- Std::binary_semaphore
- Std::counting_semaphore

binary_semaphore s{d}	counting_semaphore<LMV> s(std::ptrdiff_t d)	Gdzie “d” to wartość, która będzie przypisana licznikowi wewnętrznemu, a LMV to największa wartość, którą przyjmie nasz semafor.
release()	release(std::ptrdiff_t n = 1)	Inkrementuje wewnętrzny licznik o 1 (lub n).
acquire()	acquire()	Dekrementuje wewnętrzny licznik o 1, a gdy ten jest równy 0, to wstrzymuje dalszą pracę.
try_acquire()	try_acquire()	Gdy licznik jest większy od 0, to go dekrementuje, a gdy jest równy 0, to nie robi niczego.
try_acquire_for(const std::chrono::duration<Rep, Period>& rel_time)	try_acquire_for(const std::chrono::duration<Rep, Period>& rel_time)	Gdy licznik jest większy od 0, to go dekrementuje, a gdy jest równy 0, to czeka podany jako argument czas na zwiększenie się jego wartości.
try_acquire_until(const std::chrono::time_point<Clock, Duration>& abs_time)	try_acquire_until(const std::chrono::time_point<Clock, Duration>& abs_time)	Gdy licznik jest większy od 0, to go dekrementuje, a gdy jest równy 0, to czeka aż do podanego jako argument momentu.

Przykład

```
std::binary_semaphore   smphSignalMainToThread{ 0 },
                        smphSignalThreadToMain{ 0 };
void ThreadProc()
{
    smphSignalMainToThread.acquire();
    std::cout << "[thread] Got the signal\n";
    std::this_thread::sleep_for(3s);
    std::cout << "[thread] Send the signal\n";
    smphSignalThreadToMain.release();
}

int main()
{
    std::thread thrWorker(ThreadProc);
    std::cout << "[main] Send the signal\n";
    smphSignalMainToThread.release();
    smphSignalThreadToMain.acquire();
    std::cout << "[main] Got the signal\n";
    thrWorker.join();
    return 0;
}
```

Wynik


[main] Send the signal
[thread] Got the signal
[thread] Send the signal
[main] Got the signal

Źródła

- Cppreference.com
- See.stanford.edu
- Zadanie z PI Pani Doktor Ewy Płuciennik