O bariera este un mecanism de sincronizare care determina theadurile sa astepte pana cand un numar de threaduri ajung toate la bariera. Numarul de threaduri necesar pentru a debloca bariera se seteaza la initializarea barierei.

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

**class** my\_barrier  
{  
public:  
 my\_barrier(**int** count) : thread\_count(count), counter(0), waiting(0)  
 {}  
  
 **void** wait()  
 {  
 *//fence mechanism*  
std::unique\_lock<std::mutex> lk(m);  
 ++counter;  
 ++waiting;  
 cv.wait(lk, [&]{**return** counter >= thread\_count;});  
 cv.notify\_one();  
 --waiting;  
 **if**(waiting == 0)  
 { *//reset barrier*  
counter = 0;  
 }  
 lk.unlock();  
 }  
  
private:  
 std::mutex m;  
 std::condition\_variable cv;  
 **int** counter;  
 **int** waiting;  
 **int** thread\_count;  
};

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Si exemplu de folosire:

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

**int** thread\_waiting = 3;  
my\_barrier barrier(3);  
  
  
void func1()  
{  
 std::this\_thread::sleep\_for(std::chrono::seconds(4));  
 barrier.wait();  
 std::cout << **"I have awakened"** << std::endl;  
}  
  
void func2()  
{  
 std::this\_thread::sleep\_for(std::chrono::seconds(2));  
 barrier.wait();  
 std::cout << **"He has awakened!!"** << std::endl;  
}  
  
int test\_barrier() {  
 std::thread t1(func1);  
 std::thread t2(func2);  
 std::thread t3(func2);  
  
 t1.join();  
 t2.join();  
 t3.join();  
 std::cout << **"All have been awakened!!"** << std::endl;  
}