```cpp
#include "ThreadPool.h"

void *_executor(void *v_arg);


// Lance les différents threads de la pool
ThreadPool::ThreadPool(const int n_threads) : _threads(n_threads)
{
    sem_init(&this->_queue.sem_waiting, 0, 0);
    pthread_mutex_init(&this->_queue.mutex_waiting, NULL);

    for (int i = 0; i < n_threads; i++) {
        pthread_t t;
        pthread_create(&t, NULL, _executor, (void *) &this->_queue);

        this->_threads.push_front(t);
    }
}

ThreadPool::~ThreadPool()
{
    list<pthread_t>::iterator it;
    for (it = this->_threads.begin(); it != this->_threads.end(); it++)
        pthread_cancel(*it);

    sem_destroy(&this->_queue.sem_waiting);
    pthread_mutex_destroy(&this->_queue.mutex_waiting);
}

ThreadPool& ThreadPool::operator=(const ThreadPool& other)
{
    return *this;
}

// Attends des procédures à exécuter dans la queue.
void *_executor(void *v_arg)
{
    executor_args *arg = (executor_args *) v_arg;

    for (;;) {
        // Attends une nouvelle routine
        sem_wait(&arg->sem_waiting);

        // Retire la nouvelle routine
        pthread_mutex_lock(&arg->mutex_waiting);
        routine r = arg->waiting.front();
        arg->waiting.pop();
        pthread_mutex_unlock(&arg->mutex_waiting);

        // Exécute le routine
        r.fct(r.arg);
    }
}

// Injecte une routine dans la pool de thread.
void ThreadPool::inject(void (*fct)(void *), void *arg)
{
    routine r;
    r.fct = fct;
    r.arg = arg;

    // Rajoute la routine à la queue
    pthread_mutex_lock(&this->_queue.mutex_waiting);
    this->_queue.waiting.push(r);
    pthread_mutex_unlock(&this->_queue.mutex_waiting);

    // Signale la nouvelle routine
    sem_post(&this->_queue.sem_waiting);
}
```