```
#include <iostream>
#include <ostream>
#include <vector>
#include <limits>
#include <sstream>
#include <cmath>

#include "queue.h"

using namespace queue_simulation;

const float kInf = std::numeric_limits<float>::max();

Logger::Logger() {
    log_file_ = std::ofstream("log.txt");
}

Logger::~Logger() {
    log_file_.close();
}

void Logger::Log(std::string log) {
    log_file_ << log;
    log_file_ << std::endl;
}

std::string Simulator::GetStringVector(std::vector<float> vec) {
    std::stringstream ss;
    for(size_t i = 0; i < vec.size(); ++i)
    {
        if(i != 0)
            ss << ",";
        ss << vec[i];
    }
    std::string s = ss.str();

    return s;
}

void Simulator::Log() {
    std::stringstream system_state;
    system_state << "NEW SYSTEM STATE" << std::endl
                 << "clock: " << clock_ << std::endl
                 << "event list: " << GetStringVector(event_list_) <<
std::endl
                 << "server status: " << server_status_ << std::endl
                 << "number in queue: " << number_in_queue_ << std::endl
                 << "times of arrival: " <<
GetStringVector(arrival_times_) << std::endl
                 << "time of last event: " << last_event_time_ <<
std::endl
                 << "number serviced: " << number_serviced_ << std::endl
                 << "total delay: " << total_delay_ << std::endl
                 << "area under q(t): " << qt_area_ << std::endl
                 << "area under b(t): " << bt_area_ << std::endl
        ;

    std::string system_state_string = system_state.str();
    logger_.Log(system_state_string);
}
```

```
Simulator::Simulator(const float kLambda, const float kMu, const unsigned
kNumberServiced)
    : kLimit_(kNumberServiced), kLambda_(kLambda), kMu_(kMu) {
    event_list_.resize(2, kInf); // arrival = 0, departure = 1
    clock_ = last_event_time_ = bt_area_ = number_in_queue_ = qt_area_
        = number_serviced_ = total_delay_ = 0;
    server_status_ = false;
    wq_ = lq_ = p_ = l_ = w_ = e_s_ = 0;
}

float Simulator::GenRandomExp(float l) {
    if(l == 0) return 0.0;

    float r;
    do {
        r = std::rand();
    } while (r == 0);

    r /= (float)RAND_MAX;

    float seq1 = -1 * l;
    float seq2 = std::log(r);

    return seq1 * seq2;
}

float Simulator::GetArrivalInterval() {
    return GenRandomExp(kLambda_);
}

float Simulator::GetServiceTime() {
    return GenRandomExp(kMu_);
}

int Simulator::GetCurrentEventType() {
    float front = event_list_[0];
    float back = event_list_[1];

    if(front < back) return 0;

    // in case of equal timing return departure event
    return 1;
}

void Simulator::SetArrivalEvent() {
    float arrival_interval = GetArrivalInterval();
    event_list_[0] = arrival_interval + clock_;
}

void Simulator::SetDepartureEvent() {
    float service_time = GetServiceTime();
    event_list_[1] = service_time + clock_;

    e_s_ += service_time;
}

void Simulator::UpdateBTArea() {
    bt_area_ += clock_ - last_event_time_;
}
```

```
void Simulator::UpdateQTArea() {
    int number_in_queue = number_in_queue_;
    float interval = clock_ - last_event_time_;
    qt_area_ += number_in_queue * interval;
}

void Simulator::UpdateArrivalTimes() {
    float arrival_time = event_list_[0];
    event_list_[0] = kInf;
    arrival_times_.insert(arrival_times_.begin(), arrival_time);

    SetArrivalEvent();
}

void Simulator::UpdateTotalDelay() {
    float arrival_time = arrival_times_.back();
    arrival_times_.pop_back();
    float delay = clock_ - arrival_time;
    total_delay_ += delay;
}

void Simulator::RunSimulation() {
    if(kLimit_ == 0) return;

    // add first arrival
    float arrival_interval = GetArrivalInterval();
    event_list_[0] = arrival_interval;
    Log();

    float event_time;
    int event_type;

    // simulation
    while(number_serviced_ < kLimit_) {
        event_type = GetCurrentEventType();
        event_time = event_list_[event_type];
        last_event_time_ = clock_;
        clock_ = event_time;

        // arrival
        if(!event_type) {

            // server is idle
            if(!server_status_) {
                server_status_ = true;
                number_serviced_++;

                // update event list
                SetArrivalEvent();
                SetDepartureEvent();
            }

            // server is busy
            else {
                UpdateQTArea();
                UpdateBTArea();
                UpdateArrivalTimes();
                }
            }
```

```
            // departure
        else {
            UpdateBTArea();
            UpdateQTArea();
            if(number_in_queue_) {
                SetDepartureEvent();
                number_serviced_++;
                UpdateTotalDelay();
            }
            else {
                server_status_ = false;
                event_list_[1] = kInf;
            }
        }
        number_in_queue_ = arrival_times_.size();

        //Log();
    }

    Log();
    SetMetrics();
    LogMetrics();
}

void Simulator::SetMetrics() {
    wq_ = total_delay_ / kLimit_;
    lq_ = qt_area_ / clock_;
    p_ = bt_area_ / clock_;
    l_ = lq_ + p_;
    e_s_ = e_s_ / kLimit_;
    w_ = wq_ + e_s_;
}

void Simulator::PrintMetrics(std::string metrics) {
    std::cout << metrics;
}

void Simulator::LogMetrics() {
    std::stringstream metrics;
    metrics << "METRICS" << std::endl
            << "Wq: " << wq_ << std::endl
            << "Lq: " << lq_ << std::endl
            << "p: " << p_ << std::endl
            << "L: " << lq_ + p_ << std::endl
            << "E[s]: " << e_s_ << std::endl
            << "W: " << w_ << std::endl
        ;

    std::string metrics_string = metrics.str();
    logger_.Log(metrics_string);

    PrintMetrics(metrics_string);
}

int main() {
    const unsigned kNumberServiced = 1000000;
    const float kLambda = 1, kMu = 0.7;

    Simulator simulator(kLambda, kMu, kNumberServiced);
```

```
    simulator.RunSimulation();

    return 0;
}
```

```
NEW SYSTEM STATE
clock: 0
event list: 0.17413,3.40282e+38
server status: 0
number in queue: 0
times of arrival:
time of last event: 0
number serviced: 0
total delay: 0
area under q(t): 0
area under b(t): 0

NEW SYSTEM STATE
clock: 1.0004e+06
event list: 1.0004e+06,1.0004e+06
server status: 1
number in queue: 0
times of arrival:
time of last event: 1.0004e+06
number serviced: 1000000
total delay: 1.63053e+06
area under q(t): 1.63048e+06
area under b(t): 699297

METRICS
Wq: 1.63053
Lq: 1.62983
p: 0.69902
L: 2.32885
E[s]: 0.699333
W: 2.32987
```