Simulation - Assignment 1

Anton Roth

April 26, 2017

Introduction

The simulations has been performed with c++. For every task a brief description of the implementation is presented. I'm a PhD student in experimental nuclear physics and as with all research the results and statistical treatments and conclusions are extremely important. All possible feedback concerning this, is highly appreciated.

If not otherwise noted, the 95% confidence interval was calculated as:

$$[\bar{x} - 1.96 * \frac{\sigma}{\sqrt{N}}, \bar{x} + 1.96 * \frac{\sigma}{\sqrt{N}}]$$

Here x is the samples, $\bar{x}$ the estimated mean, σ the sample standard deviation and N the number of samples.

1 Task 1

The system with two queues was implemented with the *Event-Scheduling* approach. The states for the system were; Length of Q2, number of arrivals to Q1 and number of rejected arrivals to Q1. Four events were implemented: Arrival to Q1, Departure of Q1 (combined with arrival to Q2), Departure of Q2 and Measure.

As model verifications the following was controlled numerically and graphically:

- Inter-arrival times $\rightarrow 0$:
  - Length of Q1 $\rightarrow 10$.
  - Rejection ratio of Q1 $\rightarrow 1$.
- Inter-arrival times $\rightarrow \infty$:
  - Length of Q1 $\rightarrow 0$.
  - Rejection ratio of Q1 $\rightarrow 0$.

Measurements were made with time-differences exponentially distributed with a mean of 5 s. 10000 measurements were taken. The results of task 1 is presented in Table 1.

Table 1: Results of task 1. For the three different inter-arrival times the mean value for length of Q2 and the rejection ratio of Q1 is presented. For the mean length of Q2 the 95% confidence interval is also given. As the standard deviation of the mean estimate of the rejection is very small, it is not presented.

Inter-arrival times Q1 (s)	**Mean length Q2**	**Mean rejection ratio Q1**
1	11.50 [11.24, 11.77]	0.52
2	4.46 [4.40, 4.52]	0.069
5	0.430 [0.419, 0.441]	0

From the results of length Q2 presented in Table 1, one can with good confidence say that the mean length of the queue decreases with longer inter-arrival times, as expected. However, the sample standard deviation is on the same order as the mean which indicates that the system state varies a lot. The rejection ratios are very significant and shows an expected behaviour, i.e. the shorter inter-arrival time the higher rejection rate.

2 Task 2

The *Event-Scheduling* approach was also used in this task and the code written for task 1 was used as a template. The event structure was changed and a specific method was implemented for the addition of a job to the buffer. The following events were used: AddJobA, AddJobB, ServeJobA, ServeJobB and Measure. The states were simply the number of jobs of type A, denoted NA and B, denoted NB, in the buffer. Due to bad planning, an ugly solution was implemented for the case of adding a job from the buffer to serve (see code).

As a verification step the following was controlled:

- Delay times $\rightarrow \infty$:
  - $NA + NB \rightarrow 0$. This is due to that the serve time of A, $x_A = 0.002$ s is shorter than the average arrival time ~ 0.0067.
- Serve time for job A and B $x_A, x_B \rightarrow \infty$:
  - $NA + NB \rightarrow \infty$.

Table 2: Results of task 2 for the first three questions/simulation runs presented in the task description. For all runs the mean and the 95% confidence interval of the buffer length is presented.

"Run"	**Mean length of buffer**
1	131 [124, 137]
2	7.2 [6.7, 7.7]
3	3.6 [3.3, 3.8]

The simulation results is presented graphically in Figure 1.

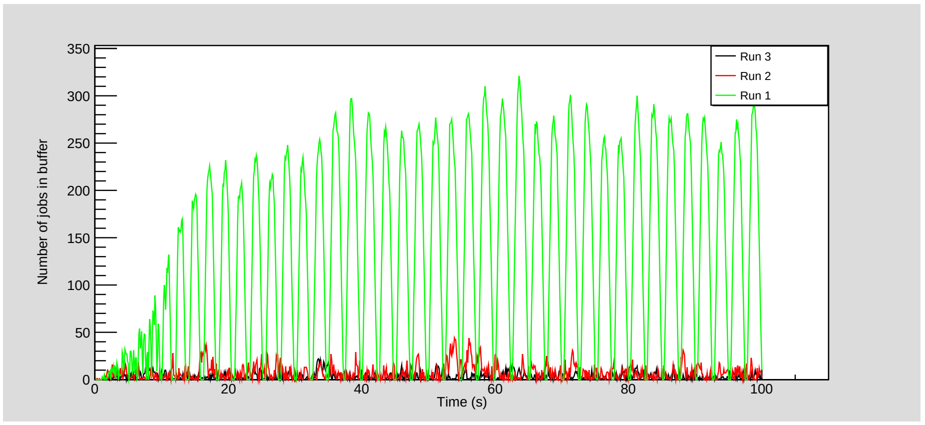

Figure 1: Result of task 2. The number of jobs in the buffer as a function of the simulation time for the three runs (indicated in the legend).

The result of run 1 and 2 differs substantially and significantly. As job B is prioritised and feeded to a system with a delay of 1 s one obtains a periodicity. Consider a constant delay of 1 s and the situation when the system starts up, the periodic behaviour can be explained with the following chain of events:

1. Job A:s are added to the buffer and served efficiently. The buffer length is kept minimal.
2. After 1 s job B:s are added to the buffer and since they are prioritised they are served immediately. The serving time is $x_B > x_A$ and together with the fact that A jobs are continuously added to the buffer this implies that the buffer length will increase.
3. After some time all B jobs have been served and A jobs are again served. The buffer length decreases.
4. $\rightarrow$ point 2.

Run 2 exhibits a weaker periodic behaviour since delay is now sampled from an exponential distribution. This has the effect that all B jobs will not arrive at the server at the same time and the buffer length will not increase as much.

The mean length of the buffer for the third run is the lowest. The buffer length no longer shows a clear periodic behaviour. As type A jobs now are prioritised and have a shorter serving time, the buffer length can efficiently be kept at a minimal. When there are no A jobs, B jobs which have a longer serving time will be processed. The pile-up in the buffer is not possible due to the efficient processing of the jobs.

It should also be mentioned that there is a “start-up” period needed for the first run (although not implemented), as it takes some time for the buffer to stabilise in its periodic behaviour.

3 Task 3

A similar system compared to in task 1 was implemented. One difference was the measure of the mean time a customer spends in the system. The time spent by every customer in the system was stored in a container and used to calculate the mean. Hence, it was not "Measured" as an event as this was considered the most efficient implementation and is not very computationally costly for the simulation.

Measurements were made with time-differences exponentially distributed with a mean of 5 s. 10000 measurements were taken. The results of task 3 is presented in Table 3.

Table 3: Results of task 3. For the three different mean arrival times the mean customer length with a 95% confidence interval and the mean queueing time for the simulation and analytical calculations are presented.

Mean arrival times (s)	**Mean customer length (simulated)**	**Mean customer length (analytically)**	**Mean queueing time (simulated)**	**Mean queueing time (analytically)**
2	1.93 [1.89, 1.97]	2	3.90 [3.86, 3.92]	4
1.5	4.14 [4.07, 4.21]	4	6.15 [6.10, 6.20]	6
1.1	18.3 [18.0, 18.5]	20	20.10 [19.97, 20.24]	22

The simulated values are somewhat close to the analytical calculations in case of 2 and 1.5 s mean arrival times. However, they are quite far from the 95% confidence interval limits. This could perhaps show some kind of uncertainty with simulations due to the random number generation. For the 1.1 s mean arrival time the simulated values deviate more from the analytical value. The reason to why, is unclear. A delayed start-up of 1000 s, chosen on the basis of the graphic evolution of the mean customer length, was tried but resulted in similar values.

4 Task 4

Task 4 was also solved with the *Event-Scheduling* approach and a similar implementation as in task 1. Three events were defined; Arrival, Depart and Measure. One system state was implemented as the number of customers, denoted NC, in the system.

As model verifications the following was controlled numerically and graphically:

- $x \to 0 : NC \to 0$
- $\lambda \to \infty : NC \to N$

4.1 Question 1, 2 & 3

The simulation measurements for the settings in questions 1, 2 and 3 is presented in Figure 2.

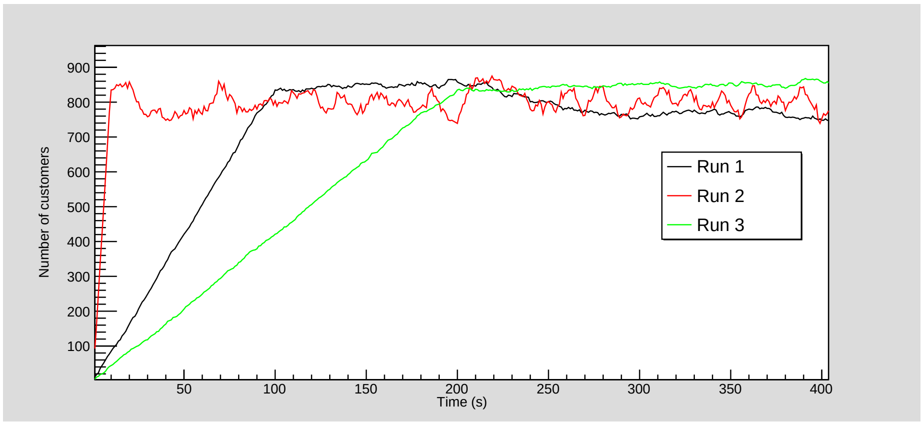

Figure 2: The number of customers as a function of simulation time is presented for the first three runs (indicated in the legend). The difference in length of the transient phase can be distinguished.

Clear from Figure 2 are different transient phases, i.e. time it takes till the system reaches an equilibrium in the total number of customers. Reading out when the measurements plane out the length of the transient phases are obtained and this is presented in Table 4.

Table 4: Length of the transient phases for task 4 and the first three questions/simulation runs.

"Run"	**Length of transient phase (s)**
1 ($x = 100, \lambda = 8$)	100
2 ($x = 10, \lambda = 80$)	10
3 ($x = 200, \lambda = 4$)	200

For these three runs the number of customers in equilibrium is the same according to Little's theorem $NC = \lambda \cdot x = 800$. The parameter that governs the length of transient phase is the rate of arrivals to the system, λ. The time it takes for the system to reach this number (assuming $x >> \frac{1}{\lambda}$) is: $t = \frac{800}{\lambda}$.

4.2 Question 4, 5 & 6

The confidence interval is calculated using the provided *MATLAB* program. In this program the system measurements are modeled as an auto-regressive process to compensate for the covariance contribution. I consider this important for the current system since in a queue a future length is always highly dependent on the current and past queue lengths. The effect on the variance from the correlation was checked with a comparison of equidistant and exponentially distributed measurement times. The covariance contribution could be observed, i.e. the computed variance with the equidistant measurement times was lower.

The length of the 95% confidence interval for the three different runs are presented in Table 5.

Table 5: Length of the 95% confidence interval for task 4 and the last three questions/simulation runs.

"Run"	**Length of 95% confidence interval.**
4 ($T = 4, M = 1000$)	1.17
5 ($T = 1, M = 4000$)	1.27
6 ($T = 4, M = 4000$)	0.64

What is very interesting from the result presented in Table 5 is that the confidence interval for run 5 is longer than that of run 4 even though the number of measurements is larger in the fifth run. The *MATLAB* modeled AR-process needs several more parameters, cf. an AR-order of 23 for run 5 and 2 for run 4. This indicates that run 5 contains a lot more information than run 4 and this is reasonable due to the 3000 more samples. My hypothesis is that run 5 catches a short periodic behaviour which run 4 does not and in turn increases the variance. The confidence intervals were also calculated for exponentially distributed measurement times and the following results were then obtained:

"Run"	**Length of 95% confidence interval.**
4 ($\exp(T = 4), M = 1000$)	2.17
5 ($\exp(T = 1), M = 4000$)	1.57
6 ($\exp(T = 4), M = 4000$)	1.00

The results presented above strengthens the hypothesis, since the periodicity in the samples should be suppressed with exponentially distributed measurements.

Run 6 has the optimal (as far as it concerns the above measurements) settings. A time of 4 s between each measurement seems to smoothen out the data and the many measurements shrinks the confidence interval, as it should.

5 Task 5

The system in this task was implemented with the *Process Interaction* method. It took quite some time to implement.

Three different processes, i.e. classes, were implemented; Generator, Queue and Measure. The interaction between the processes was implemented with a Signal class which comprised a SignalType, a receiving process Destination, and ArrivalTime, plus an optional parameter representing the measured values sent to the Measure process. A helper class ProcessList holds all Processes and controlls the Signal which is to be treated. A base class LoadDistr with derived classes RandomLoad, RobinLoad and SmallestQueLoad were also implemented. The Generator process possesses an object of type LoadDistr which has a function GetQ which is invoked when Queue is chosen.

The system was first tried on just one Queue and verified with Little's theorem. In this process it was noted that if the mean arrival time was smaller than the mean service time, the system diverged. It was observed that for mean inter-arrival times $\rightarrow$ mean service time a mean queue length longer than the value from Little's theorem was always obtained. However, for larger inter-arrival times the queue length agreed with Little's theorem. Why this is the case is still unclear. The chain of signals have been closely controlled and these work as they are intended. The sampling from the uniform distribution and the exponential distribution has also been verified to work.

Even though the system might not work as it should the full implementation was made. The system behaved the same with 5 queues and the load distribution as discussed in the above section. Hence, with a mean arrival time of 0.12 seconds an obtained queue length quite longer than Little's theorem indicated. Although, the system does not seem to work correctly quantitatively it should qualititatively. The system was simulated for 100 000 s and the results for the different arrival times and load distributions are presented in Table 7.

Table 6: Results of task 5. For the three different inter-arrival times the mean value and its 95% confidence interval are presented for all load distributions and mean arrival times.

Inter-arrival times (s)	**Random load distribution**	**Robin load distribution**	**Smallest queue distribution**
0.11	44.9 [44.6, 45.2]	26.3 [25.9, 26.6]	11.8 [11.7, 12.0]
0.15	9.26 [9.20, 9.32]	5.98 [5.89, 6.06]	3.97 [3.94, 4.00]
2	0.257 [0.250, 0.263]	0.256 [0.249, 0.263]	0.252 [0.245, 0.258]

From the results presented in Table 6 it is clear that the smallest queue distribution is has the lowest average queue length for mean arrival times 0.11 and 0.15 s. For the same mean arrival time, it can also be seen that the Robin load is the second best. For the longest mean arrival time all confidence interval overlap which indicates that no load distribution is better than the other. Furthermore, the shorter arrival times the relative improvement possible by changing to the smallest queue distribution increases (e.g. $\frac{26.3}{11.8} > \frac{5.98}{3.97}$).

6 Task 6

The system was implemented with a simplified *Event-Scheduling* approach. Two events; Arrival and Depart was implemented. The system state was the number of prescriptions to be filled in. New arrivals were generated if the arrival time was before the closing time. The stopping condition for a working day's simulation was an empty event list and 1000 days were simulated. The event time of each day's last departure was measured and stored in a container. The mean of the container elements was the average time his work will finish every day. The time difference of the arrival and departure of each prescription was also stored in a container and used to calculate the average time it takes to fill a prescription. The results of task 6 is presented in Table 7.

Table 7: Results of task 6. The average time the work finishes and the average time it takes to fill a prescription is presented with the 95% confidence interval.

Average time work finishes	Average time it takes to fill prescription
17:23:30 [17:22:34, 17:24:26]	25.1 [25.0, 25.2] min

7 Task 7

The system in task 7 was implemented with a small scale *Event-Scheduling* approach. First the life time for each of the components were generated and stored in an event list, then the event list was stepped through in chronological order and the time at which all components were "dead" was stored in a container. The mean of this container, which is the average life time of the system, is the result presented in Table 8.

Table 8: The result of task 7. The average life time is presented with its 95% confidence interval.

Average life time of the system
$3.67[3.61, 3.73]$

Code

Below the c++-code used for the assignement is presented. Header files and Makefiles have been left out. I would strongly recommend to study the source code in the attached .zip file instead of trying to read here. Also, note that the framework ROOT [`https://root.cern.ch/`] has been used to visualise the data.

7.1 Task 1

File: eventandstate.cc

```
#include "eventandstate.h"

#include <iostream>
#include <algorithm>
#include <math.h>

#include "TString.h"
#include "TFile.h"
#include "TGraph.h"
```

```
#include "TCanvas.h"
#include "TAxis.h"
#include "TColor.h"
#include "TLegend.h"
#include "TPaveText.h"
#include "TLatex.h"

using namespace std;

void EventList::InsertEvent(int type, double time) {
        Event e(type, time);
        if(event_list.size() == 0) {
                event_list.push_back(e);
                return;
        }

        auto it = event_list.begin();
        while(it != event_list.end() && e.eventtime > it
            ↪ ->eventtime) {
                ++it;
        }

        it = event_list.insert(it, e);
}

State::State(default_random_engine& re, double dt) : LQ1
    ↪ (0), LQ2(0), nbr_arrivalsQ1(0), nbr_rejectedQ1(0),
    ↪ nbr_measurements(0), rnd_engine(re), dtQ1(dt) {

}

void State::ProcessEvent(EventList& el) {
        Event e = el.event_list[0];
        switch (e.eventtype){
                case Event::ArrivalQ1:
                        cout << "ArrivalQ1:␣" << e <<
                            ↪ endl;
                        if(LQ1 < 10) {
                                ++LQ1;
                                ++nbr_arrivalsQ1;
                        }
                        else {
                                ++nbr_arrivalsQ1;
                                ++nbr_rejectedQ1;
                        }
                        el.InsertEvent(Event::ArrivalQ1,
                            ↪ e.eventtime + dtQ1);
                        if(LQ1 == 1) el.InsertEvent(Event
                            ↪ ::DepartQ1, e.eventtime +
                            ↪ get_exp_time(rnd_engine,
                            ↪ 2.1));
                        break;
                case Event::DepartQ1:
                        cout << "DepartQ1: " << e << endl
                            ↪ ;
                        --LQ1;
                        ++LQ2;
                        if(LQ1 > 0) el.InsertEvent(Event
                            ↪ ::DepartQ1, e.eventtime +
                            ↪ get_exp_time(rnd_engine,
                            ↪ 2.1));
                        if(LQ2 == 1) el.InsertEvent(Event
                            ↪ ::DepartQ2, e.eventtime +
                            ↪ 2);
                        break;
                case Event::DepartQ2:
                        cout << "DepartQ2: " << e << endl
                            ↪ ;
                        --LQ2;
                        if(LQ2 > 0) el.InsertEvent(Event
                            ↪ ::DepartQ2, e.eventtime +
                            ↪ 2);
                        break;
                case Event::Measure:
                        cout << "Measuring: " << LQ1 << "
                            ↪ , "<< LQ2 << ", " << e.
                            ↪ eventtime << endl;
                        ++nbr_measurements;
                        el.InsertEvent(Event::Measure, e.
                            ↪ eventtime + get_exp_time(
                            ↪ rnd_engine, 5));
                        v_LQ1.push_back(LQ1);
                        v_LQ2.push_back(LQ2);
                        v_mean.push_back(calc_mean(v_LQ2)
                            ↪ );
                        v_var.push_back(calc_stddev(v_LQ2
                            ↪ ));
                        v_time.push_back(e.eventtime);
                        if(nbr_arrivalsQ1 != 0) {
                                v_rej_ratio.push_back(
                                    ↪ static_cast<double
                                    ↪ >(nbr_rejectedQ1)/
```

```
                                        ↪ static_cast<double
                                        ↪ >(nbr_arrivalsQ1));
                        }
                        else v_rej_ratio.push_back(0);
                        break;
        }
}
void State::Write(string s) {
        cout << "Writing to file task1.root" << endl;
        TFile* f_out = new TFile(TString(s), "RECREATE");

        TGraph* g;
  TCanvas* C = new TCanvas();

  //double marker_size[runs] = {1, 2, 1, 2, 2};
  //double marker_style[runs] = {21, 21, 22, 22, 20};
  //red, green, blue, cyan, black
  //TColor marker_colour[runs] = {TColor(1,0,0), TColor
      ↪ (0,1,0), TColor(0,0,1), TColor(0,1,1), TColor
      ↪ (0,0,0)};
  TLegend* leg = new TLegend(0.7,0.7,0.55,0.9);
  //leg->SetHeader("Collimator","C"); // option "C"
      ↪ allows to center the header, WHY DON'T WORK?
  //leg->SetHeader("The Legend Title",""); // option "C"
      ↪ allows to center the header
  //TString legend_labels[runs] = {TString("Simple, 1 mm
      ↪ diameter"), TString("Simple, 1.5 mm diameter"),
      ↪ TString("Integrated cones (i)"),TString("
      ↪ Integrated cones (ii)"), TString("Integrated
      ↪ cylinders")};

        TString s_x[2] = {"Time (s)", "Time (s)"};
        TString s_y[5] = {"Length Q1", "Length Q2", "
            ↪ Rejection probability in Q1", "Mean LQ2", "
            ↪ StdDev LQ2"};

        g = new TGraph(v_LQ2.size(), &(v_time[0]), &(
            ↪ v_LQ1[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[0]);
```

```
g->GetXaxis()->SetTitle(s_x[0]);
g->GetYaxis()->CenterTitle();
g->GetXaxis()->CenterTitle();
g->Write();

g = new TGraph(v_LQ2.size(), &(v_time[0]), &(
    ↪ v_LQ2[0]));
g->Draw("ACP"); //For the first one, one needs to
    ↪  draw axis with "A". Option "SAME" is not
    ↪ needed with TGraph!
//Axis objects for TGraph are created after it
    ↪ has been drawn, thus they need to be
    ↪ defined here.
g->SetTitle("");
g->GetYaxis()->SetTitle(s_y[1]);
g->GetXaxis()->SetTitle(s_x[1]);
g->GetYaxis()->CenterTitle();
g->GetXaxis()->CenterTitle();
g->Write();

g = new TGraph(v_rej_ratio.size(), &(v_time[0]),
    ↪ &(v_rej_ratio[0]));
g->Draw("ACP"); //For the first one, one needs to
    ↪  draw axis with "A". Option "SAME" is not
    ↪ needed with TGraph!
//Axis objects for TGraph are created after it
    ↪ has been drawn, thus they need to be
    ↪ defined here.
g->SetTitle("");
g->GetYaxis()->SetTitle(s_y[2]);
g->GetXaxis()->SetTitle(s_x[1]);
g->GetYaxis()->CenterTitle();
g->GetXaxis()->CenterTitle();
g->Write();

g = new TGraph(v_mean.size(), &(v_time[0]), &(
    ↪ v_mean[0]));
g->Draw("ACP"); //For the first one, one needs to
    ↪  draw axis with "A". Option "SAME" is not
    ↪ needed with TGraph!
//Axis objects for TGraph are created after it
    ↪ has been drawn, thus they need to be
    ↪ defined here.
g->SetTitle("");
g->GetYaxis()->SetTitle(s_y[3]);
g->GetXaxis()->SetTitle(s_x[1]);
```

```
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_var.size(), &(v_time[0]), &(
            ↪ v_var[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[4]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();


        f_out->Close();


}

double get_exp_time(default_random_engine& rnd, double mu
    ↪ ) {
        exponential_distribution<double> dist_exp(1/mu);
        double rand = dist_exp(rnd);
        cout << "rnd␣=␣" << rand << endl;
        return rand;
}

double calc_mean(vector<double>& v) {
        double sum = accumulate(v.begin(), v.end(), 0.0);
        return sum/v.size();
}

double calc_stddev(vector<double>& v) {
        double mean = calc_mean(v);
        vector<double> diff(v.size());
        transform(v.begin(), v.end(), diff.begin(), [mean
            ↪ ](double x) { return x - mean; });
        double sq_sum = inner_product(diff.begin(), diff.
            ↪ end(), diff.begin(), 0.0);
        double stdev = sqrt(sq_sum / v.size());
        return stdev;
}

ostream& operator<<(ostream& os, Event e) {
        os << "Event:␣" << e.eventtype << ",␣" << e.
            ↪ eventtime << "␣s";
        return os;
}
```

File: main.cc

```
#include "eventandstate.h"

#include <iostream>
#include <fstream>

using namespace std;

void stats(ofstream& ofs, vector<double> v) {
        double mean = calc_mean(v);
        double stddev = calc_stddev(v);
        double stddev_mean = stddev/sqrt(v.size());
        ofs << "mean␣=␣" << mean << endl;
        ofs << "stddev␣of␣mean␣estimate␣=␣" <<
            ↪ stddev_mean << endl;
        ofs << "95%␣conf␣int␣=␣" << mean - 1.96*
            ↪ stddev_mean << ",␣" << mean + 1.96*
            ↪ stddev_mean << endl;
        ofs << "90%␣conf␣int␣=␣" << mean - 1.645*
            ↪ stddev_mean << ",␣" << mean + 1.645*
            ↪ stddev_mean << endl;
        ofs << "99%␣conf␣int␣=␣" << mean - 2.576*
            ↪ stddev_mean << ",␣" << mean + 2.576*
            ↪ stddev_mean << endl;
        cout << endl;
}

int main() {
        cout << "Running␣task␣1␣" << endl;
        ofstream f_res("res_task1.txt");

        vector<double> dtQ1 = {0.1, 1, 2, 5, 20};
        //vector<double> dtQ1 = {1, 2, 5};

        for(auto& dt : dtQ1) {
                EventList el;
                int seed = 0;
                default_random_engine rnd_engine(seed);


                //Kick-start:
                el.InsertEvent(Event::ArrivalQ1, dt);
                el.InsertEvent(Event::Measure,
                    ↪ get_exp_time(rnd_engine, 5));

                State s(rnd_engine, dt);

                while(s.nbr_measurements < 10000) {
                        //cout << "LQ1 = " << s.LQ1 <<
                            ↪ endl;
                        if(s.LQ1 > 10) {
                                exit(1);
                        }
                        s.ProcessEvent(el);
                        auto it_temp = el.event_list.
                            ↪ erase(el.event_list.begin()
                            ↪ );
                        //for(auto& e : el.event_list)
                            ↪ cout << e << endl;
                }

                s.Write("task1_"+to_string(dt)+".root");

                f_res << "Results for dtQ1 = " << dt << "
                    ↪  LQ2 " << endl;
                stats(f_res, s.v_LQ2);
                f_res << "Results for dtQ1 = " << dt << "
                    ↪  rejection rate " << endl;
                stats(f_res, s.v_rej_ratio);
        }



}
```

7.2 Task 2

File: eventandstate.cc

```
#include "eventandstate.h"

#include <iostream>
```

```
#include <algorithm>
#include <math.h>

#include "TString.h"
#include "TFile.h"
#include "TGraph.h"
#include "TCanvas.h"
#include "TAxis.h"
#include "TColor.h"
#include "TLegend.h"
#include "TPaveText.h"
#include "TLatex.h"

using namespace std;

void EventList::InsertEvent(int type, double time) {
        Event e(type, time);
        if(event_list.size() == 0) {
                event_list.push_back(e);
                return;
        }

        auto it = event_list.begin();
        while(it != event_list.end() && e.eventtime >= it
            ↪ ->eventtime) {
                ++it;
        }

        it = event_list.insert(it, e);
}

State::State(default_random_engine& re, bool is_B,
    ↪ unsigned int c) : NA(0), NB(0), nbr_measurements(0)
    ↪ , rnd_engine(re), is_B_priority(is_B),
    ↪ is_server_busy(false), count(c) {

        xA = 0.002;
        xB = 0.004;
        d = 1;
        lambda = 150;

}

void State::ProcessEvent(EventList& el) {
        Event e = el.event_list[0];
        cout << "Processing event: " << endl;
    switch (e.eventtype){
        case Event::AddJobA:
            cout << "AddJobA␣" << e << endl;
            el.InsertEvent(Event::AddJobA, e.
               ↪ eventtime + get_exp_time(
               ↪ rnd_engine, 1./lambda));
            //If buffer empty and server not
               ↪ busy, need to trigger serve
               ↪ .
            if(v_Buffer.size() == 0 && !
               ↪ is_server_busy) {
                el.InsertEvent(Event::
                   ↪ ServeJobA, e.
                   ↪ eventtime + xA);
                is_server_busy = true;
                break;
            }
            AddJobToBuffer(e);
            ++NA;
            break;
        case Event::AddJobB:
            cout << "AddJobB␣" << e << endl;
            //If buffer empty, need to
               ↪ trigger serve.
            if(v_Buffer.size() == 0 && !
               ↪ is_server_busy) {
                el.InsertEvent(Event::
                   ↪ ServeJobB, e.
                   ↪ eventtime + xB);
                is_server_busy = true;
                break;
            }
            AddJobToBuffer(e);
            ++NB;
            break;
        case Event::ServeJobA:
            cout << "ServeJobA:␣" << e <<
               ↪ endl;
            if(v_Buffer.size() != 0) {
                //This if statement is an
                   ↪  ugly solution, due
                   ↪  to bad planning
                if(v_Buffer[0].eventtype
                   ↪ == Event::AddJobB)
                   ↪ {
                    cout << "
                    ↪ Processing␣
                    ↪ Job␣B" <<
                    ↪ endl;
                      el.
                          ↪ InsertEvent
                          ↪ (
                          ↪ Event
                          ↪ ::
                          ↪ ServeJobB
                          ↪ , e
                          ↪ .
                          ↪ eventtime
                          ↪  +
                          ↪ xB)
                          ↪ ;
                      auto it =
                          ↪
                          ↪ v_Buffer
                          ↪ .
                          ↪ erase
                          ↪ (
                          ↪ v_Buffer
                          ↪ .
                          ↪ begin
                          ↪ ())
                          ↪ ;
                      --NB;
                }
                else {
                      cout << "
                          ↪ Processing
                          ↪ ␣
                          ↪ Job
                          ↪ ␣A"
                          ↪  <<
                          ↪
                          ↪ endl
                          ↪ ;
                      el.
                          ↪ InsertEvent
                          ↪ (
                          ↪ Event
                          ↪ ::
                          ↪ ServeJobA
                          ↪ , e
                          ↪ .
```

```
    ↪ eventtime
    ↪  +
    ↪ xA)
    ↪ ;
auto it =
    ↪
    ↪ v_Buffer
    ↪ .
    ↪ erase
    ↪ (
    ↪ v_Buffer
    ↪ .
    ↪ begin
    ↪ ())
    ↪ ;
--NA;
if(count
    ↪ ==
    ↪ 2)
    ↪ {
        el
            ↪ .
            ↪ InsertEvent
            ↪ (
            ↪ Event
            ↪ ::
            ↪ AddJobB
            ↪ ,
            ↪
            ↪ e
            ↪ .
            ↪ eventtime
            ↪
            ↪ +
            ↪
            ↪ get_exp_time
            ↪ (
            ↪ rnd_engine
            ↪ ,
            ↪
            ↪ 1)
            ↪ )
            ↪ ;
            ↪
}
else el.
```

```
                                        ↪ InsertEvent
                                        ↪ (
                                        ↪ Event
                                        ↪ ::
                                        ↪ AddJobB
                                        ↪ , e
                                        ↪ .
                                        ↪ eventtime
                                        ↪  +
                                        ↪ d);
                                }
                                is_server_busy =
                                    ↪ true;
                        }
                        else {
                                is_server_busy = false;
                                if(count == 2) {
                                        el.InsertEvent(
                                            ↪ Event::
                                            ↪ AddJobB, e.
                                            ↪ eventtime +
                                            ↪
                                            ↪ get_exp_time
                                            ↪ (rnd_engine
                                            ↪ , 1));
                                }
                                else el.InsertEvent(Event
                                    ↪ ::AddJobB, e.
                                    ↪ eventtime + d);
                        }
                        break;
                case Event::ServeJobB:
                        cout << "ServeJobB: " << e <<
                            ↪ endl;
                        if(v_Buffer.size() != 0) {
                                //This if statement is an
                                    ↪  ugly solution, due
                                    ↪  to bad planning
                                if(v_Buffer[0].eventtype
                                    ↪ == Event::AddJobB)
                                    ↪ {
                                                el.
                                                    ↪ InsertEvent
                                                    ↪ (
                                                    ↪ Event
                                                    ↪ ::
```

```
                                                ↪ ServeJobB
                                                ↪ , e
                                                ↪ .
                                                ↪ eventtime
                                                ↪  +
                                                ↪ xB)
                                                ↪ ;
                                            auto it =
                                                ↪
                                                ↪ v_Buffer
                                                ↪ .
                                                ↪ erase
                                                ↪ (
                                                ↪ v_Buffer
                                                ↪ .
                                                ↪ begin
                                                ↪ ())
                                                ↪ ;
                                            --NB;
                                }
                                else {
                                            el.
                                                ↪ InsertEvent
                                                ↪ (
                                                ↪ Event
                                                ↪ ::
                                                ↪ ServeJobA
                                                ↪ , e
                                                ↪ .
                                                ↪ eventtime
                                                ↪  +
                                                ↪ xA)
                                                ↪ ;
                                            auto it =
                                                ↪
                                                ↪ v_Buffer
                                                ↪ .
                                                ↪ erase
                                                ↪ (
                                                ↪ v_Buffer
                                                ↪ .
                                                ↪ begin
                                                ↪ ())
                                                ↪ ;
                                            --NA;
                                if(count == 2) {
```

```
                                                el.
                                                    ↪ InsertEvent
                                                    ↪ (
                                                    ↪ Event
                                                    ↪ ::
                                                    ↪ AddJobB
                                                    ↪ , e
                                                    ↪ .
                                                    ↪ eventtime
                                                    ↪  +
                                                    ↪ get_exp_time
                                                    ↪ (
                                                    ↪ rnd_engine
                                                    ↪ ,
                                                    ↪ 1))
                                                    ↪ ;
                                        }
                                        else el.
                                            ↪ InsertEvent
                                            ↪ (Event::
                                            ↪ AddJobB, e.
                                            ↪ eventtime +
                                            ↪  d);
                                        }
                                        is_server_busy =
                                            ↪ true;
                        }
                        else {
                                is_server_busy = false;
                        }
                        break;
                case Event::Measure:
                        cout << "Measuring:␣" << NA << ",
                            ↪ ␣"<< NB << ",␣" << e.
                            ↪ eventtime << endl;
                        ++nbr_measurements;
                        el.InsertEvent(Event::Measure, e.
                            ↪ eventtime + 0.1);
                        v_NA.push_back(NA);
                        v_NB.push_back(NB);
                        v_NAB.push_back(v_Buffer.size());
                        v_mean.push_back(calc_mean(v_NAB)
                            ↪ );
                        v_var.push_back(calc_stddev(v_NAB
                            ↪ ));
                        v_time.push_back(e.eventtime);
```

```
                                break;
            }
}

void State::AddJobToBuffer(Event e) {
            cout << "Adding to buffer Job " << e.eventtype <<
                ↪  endl;
            if(v_Buffer.size() == 0) {
                        v_Buffer.push_back(e);
                        return;
            }
            if( (e.eventtype == Event::AddJobA && !
                ↪ is_B_priority) || (e.eventtype == Event::
                ↪ AddJobB && is_B_priority)) {
                        auto it = find_if(v_Buffer.begin(),
                            ↪ v_Buffer.end(), [e] (const Event& b
                            ↪ ) {return b.eventtype != e.
                            ↪ eventtype;});
                        v_Buffer.insert(it, e);
            }
            else v_Buffer.push_back(e);

}

void State::Write(string s) {
            cout << "Writing to file task1.root" << endl;
            TFile* f_out = new TFile(TString(s), "RECREATE");

            TGraph* g;
   TCanvas* C = new TCanvas();

   //double marker_size[runs] = {1, 2, 1, 2, 2};
   //double marker_style[runs] = {21, 21, 22, 22, 20};
   //red, green, blue, cyan, black
   //TColor marker_colour[runs] = {TColor(1,0,0), TColor
       ↪ (0,1,0), TColor(0,0,1), TColor(0,1,1), TColor
       ↪ (0,0,0)};
   TLegend* leg = new TLegend(0.7,0.7,0.55,0.9);
   //leg->SetHeader("Collimator","C"); // option "C"
       ↪ allows to center the header, WHY DON'T WORK?
   //leg->SetHeader("The Legend Title",""); // option "C"
       ↪ allows to center the header
   //TString legend_labels[runs] = {TString("Simple, 1 mm
       ↪ diameter"), TString("Simple, 1.5 mm diameter"),
       ↪ TString("Integrated cones (i)"),TString("
       ↪ Integrated cones (ii)"), TString("Integrated
↪ cylinders")};

    TString s_x[2] = {"Time␣(s)", "Time␣(s)"};
    TString s_y[5] = {"NAB", "NA", "NB", "Mean_NAB",
        ↪ "StdDev_NAB"};

    g = new TGraph(v_NAB.size(), &(v_time[0]), &(
        ↪ v_NAB[0]));
    g->Draw("ACP"); //For the first one, one needs to
        ↪  draw axis with "A". Option "SAME" is not
        ↪ needed with TGraph!
    //Axis objects for TGraph are created after it
        ↪ has been drawn, thus they need to be
        ↪ defined here.
    g->SetTitle("");
    g->GetYaxis()->SetTitle(s_y[0]);
    g->GetXaxis()->SetTitle(s_x[0]);
    g->GetYaxis()->CenterTitle();
    g->GetXaxis()->CenterTitle();
    g->Write();

    g = new TGraph(v_NAB.size(), &(v_time[0]), &(v_NA
        ↪ [0]));
    g->Draw("ACP"); //For the first one, one needs to
        ↪  draw axis with "A". Option "SAME" is not
        ↪ needed with TGraph!
    //Axis objects for TGraph are created after it
        ↪ has been drawn, thus they need to be
        ↪ defined here.
    g->SetTitle("");
    g->GetYaxis()->SetTitle(s_y[1]);
    g->GetXaxis()->SetTitle(s_x[1]);
    g->GetYaxis()->CenterTitle();
    g->GetXaxis()->CenterTitle();
    g->Write();

    g = new TGraph(v_NB.size(), &(v_time[0]), &(v_NB
        ↪ [0]));
    g->Draw("ACP"); //For the first one, one needs to
        ↪  draw axis with "A". Option "SAME" is not
        ↪ needed with TGraph!
    //Axis objects for TGraph are created after it
        ↪ has been drawn, thus they need to be
        ↪ defined here.
    g->SetTitle("");
    g->GetYaxis()->SetTitle(s_y[2]);
```

```
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_mean.size(), &(v_time[0]), &(
            ↪ v_mean[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[3]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_var.size(), &(v_time[0]), &(
            ↪ v_var[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[4]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();


        f_out->Close();


}

double get_exp_time(default_random_engine& rnd, double mu
    ↪ ) {
        exponential_distribution<double> dist_exp(1/mu);
        double rand = dist_exp(rnd);
        return rand;
```

```
}

double calc_mean(vector<double>& v) {
        double sum = accumulate(v.begin(), v.end(), 0.0);
        return sum/v.size();
}

double calc_stddev(vector<double>& v) {
        double mean = calc_mean(v);
        vector<double> diff(v.size());
        transform(v.begin(), v.end(), diff.begin(), [mean
            ↪ ](double x) { return x - mean; });
        double sq_sum = inner_product(diff.begin(), diff.
            ↪ end(), diff.begin(), 0.0);
        double stdev = sqrt(sq_sum / v.size());
        return stdev;
}

ostream& operator<<(ostream& os, Event e) {
        os << "Event:␣" << e.eventtype << ",␣" << e.
            ↪ eventtime << "␣s";
        return os;
}
```

File: main.cc

```
#include "eventandstate.h"

#include <iostream>
#include <iterator>
#include <fstream>

using namespace std;

void stats(ofstream& ofs, vector<double> v) {
        double mean = calc_mean(v);
        double stddev = calc_stddev(v);
        double stddev_mean = stddev/sqrt(v.size());
        ofs << "mean␣=␣" << mean << endl;
        ofs << "stddev␣of␣mean␣estimate␣=␣" <<
            ↪ stddev_mean << endl;
        ofs << "95%␣conf␣int␣=␣" << mean - 1.96*
            ↪ stddev_mean << ",␣" << mean + 1.96*
            ↪ stddev_mean << endl;
        ofs << "90%␣conf␣int␣=␣" << mean - 1.645*
            ↪ stddev_mean << ",␣" << mean + 1.645*
            ↪ stddev_mean << endl;
        ofs << "99%␣conf␣int␣=␣" << mean - 2.576*
            ↪ stddev_mean << ",␣" << mean + 2.576*
            ↪ stddev_mean << endl;
        cout << endl;
}

int main() {
        cout << "Running␣task␣2␣" << endl;
        ofstream f_res("res_task2.txt");

        vector<bool> is_B = {true, false, true};

        int count = 0;
        for(auto B : is_B) {
                EventList el;
                int seed = 0;
                default_random_engine rnd_engine(seed);


                //Kick-start:
                double t_1st_job = get_exp_time(
                    ↪ rnd_engine, 1./150);
                el.InsertEvent(Event::AddJobA, t_1st_job)
                    ↪ ;
                el.InsertEvent(Event::Measure, 0.1);

                State s(rnd_engine, B, count);

                while(s.nbr_measurements < 1000) {
                        cout << "NA␣=␣" << s.NA << endl;
                        //if(s.NA > 2000) exit(1);
                        s.ProcessEvent(el);
                        auto it_temp = el.event_list.
                            ↪ erase(el.event_list.begin()
                            ↪ );
                        /*
                        for(auto& e : s.v_Buffer){
                                cout << "Buffer: ";
                                cout << e << endl;
                        }
                        */

                        copy(el.event_list.begin(), el.
                            ↪ event_list.begin()+3,
                            ↪ ostream_iterator<Event>(
                            ↪ cout, "\n"));
                }

                if(count == 2) s.Write("task2_"+to_string
                    ↪ (B)+"_Exp.root");
                else s.Write("task2_"+to_string(B)+".root
                    ↪ ");

                f_res << "Results for IsBPriority? = " <<
                    ↪ B << " and NAB" << endl;
                stats(f_res, s.v_NAB);
                f_res << "Results for IsBPriority? = " <<
                    ↪ B << " and NA" << endl;
                stats(f_res, s.v_NA);
                f_res << "Results for IsBPriority? = " <<
                    ↪ B << " and NB" << endl;
                stats(f_res, s.v_NB);

                cout << "
                    ↪ ##########################################################
                    ↪ " << endl;

                ++count;
        }



}
```

7.3 Task 3

File: eventandstate.cc

```
#include "eventandstate.h"

#include <iostream>
#include <algorithm>
#include <math.h>

#include "TString.h"
#include "TFile.h"
#include "TGraph.h"
#include "TCanvas.h"
#include "TAxis.h"
#include "TColor.h"
#include "TLegend.h"
```

```
#include "TPaveText.h"
#include "TLatex.h"

using namespace std;

void EventList::InsertEvent(int type, double time) {
        Event e(type, time);
        if(event_list.size() == 0) {
                event_list.push_back(e);
                return;
        }

        auto it = event_list.begin();
        while(it != event_list.end() && e.eventtime > it
            ↪ ->eventtime) {
                ++it;
        }

        it = event_list.insert(it, e);
}

State::State(default_random_engine& re, double dt) : LQ1
    ↪ (0), LQ2(0), nbr_arrivalsQ1(0), nbr_rejectedQ1(0),
    ↪ nbr_measurements(0), rnd_engine(re), dtQ1(dt) {

}

void State::ProcessEvent(EventList& el) {
        Event e = el.event_list[0];
        switch (e.eventtype){
                case Event::ArrivalQ1:
                        cout << "ArrivalQ1:␣" << e <<
                            ↪ endl;
                        ++LQ1;
                        el.InsertEvent(Event::ArrivalQ1,
                            ↪ e.eventtime + get_exp_time(
                            ↪ rnd_engine, dtQ1));
                        if(LQ1 == 1) el.InsertEvent(Event
                            ↪ ::DepartQ1, e.eventtime +
                            ↪ get_exp_time(rnd_engine, 1)
                            ↪ );
                        v_in_Q.push_back(e.eventtime);
                        break;
                case Event::DepartQ1:
                        cout << "DepartQ1:␣" << e << endl
                            ↪ ;
                        --LQ1;
                        ++LQ2;
                        if(LQ1 > 0) el.InsertEvent(Event
                            ↪ ::DepartQ1, e.eventtime +
                            ↪ get_exp_time(rnd_engine, 1)
                            ↪ );
                        if(LQ2 == 1) el.InsertEvent(Event
                            ↪ ::DepartQ2, e.eventtime +
                            ↪ get_exp_time(rnd_engine, 1)
                            ↪ );
                        break;
                case Event::DepartQ2:
                        cout << "DepartQ2:␣" << e << endl
                            ↪ ;
                        --LQ2;
                        if(LQ2 > 0) el.InsertEvent(Event
                            ↪ ::DepartQ2, e.eventtime +
                            ↪ get_exp_time(rnd_engine, 1)
                            ↪ );
                        v_mean_time.push_back(e.eventtime
                            ↪  - v_in_Q[0]);
                        v_in_Q.erase(v_in_Q.begin());
                        break;
                case Event::Measure:
                        cout << "Measuring:␣" << LQ1 << "
                            ↪ ,␣"<< LQ2 << ",␣" << e.
                            ↪ eventtime << endl;
                        ++nbr_measurements;
                        el.InsertEvent(Event::Measure, e.
                            ↪ eventtime + get_exp_time(
                            ↪ rnd_engine, 5));
                        v_LQ1.push_back(LQ1+LQ2);
                        v_LQ2.push_back(LQ2);
                        v_mean.push_back(calc_mean(v_LQ2)
                            ↪ );
                        v_var.push_back(calc_stddev(v_LQ2
                            ↪ ));
                        v_time.push_back(e.eventtime);
                        break;
        }
}
void State::Write(string s) {
        cout << "Writing␣to␣file␣task3.root" << endl;
        TFile* f_out = new TFile(TString(s), "RECREATE");

        TGraph* g;
```

```
TCanvas* C = new TCanvas();

//double marker_size[runs] = {1, 2, 1, 2, 2};
//double marker_style[runs] = {21, 21, 22, 22, 20};
//red, green, blue, cyan, black
//TColor marker_colour[runs] = {TColor(1,0,0), TColor
    ↪ (0,1,0), TColor(0,0,1), TColor(0,1,1), TColor
    ↪ (0,0,0)};
TLegend* leg = new TLegend(0.7,0.7,0.55,0.9);
//leg->SetHeader("Collimator","C"); // option "C"
    ↪ allows to center the header, WHY DON'T WORK?
//leg->SetHeader("The Legend Title",""); // option "C"
    ↪ allows to center the header
//TString legend_labels[runs] = {TString("Simple, 1 mm
    ↪ diameter"), TString("Simple, 1.5 mm diameter"),
    ↪ TString("Integrated cones (i)"),TString("
    ↪ Integrated cones (ii)"), TString("Integrated
    ↪ cylinders")};

        TString s_x[2] = {"Time␣(s)", "Time␣(s)"};
        TString s_y[4] = {"Length␣Q1", "Length␣Q2", "Mean
            ↪ ␣LQ2", "StdDev␣LQ2"};

        g = new TGraph(v_LQ2.size(), &(v_time[0]), &(
            ↪ v_LQ1[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[0]);
        g->GetXaxis()->SetTitle(s_x[0]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_LQ2.size(), &(v_time[0]), &(
            ↪ v_LQ2[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[1]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_mean.size(), &(v_time[0]), &(
            ↪ v_mean[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[2]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_var.size(), &(v_time[0]), &(
            ↪ v_var[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[3]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();


        f_out->Close();


}

double get_exp_time(default_random_engine& rnd, double mu
    ↪ ) {
        exponential_distribution<double> dist_exp(1/mu);
        double rand = dist_exp(rnd);
        cout << "rnd = " << rand << endl;
        return rand;
}

double calc_mean(vector<double>& v) {
        double sum = accumulate(v.begin(), v.end(), 0.0);
        return sum/v.size();
}

double calc_stddev(vector<double>& v) {
        double mean = calc_mean(v);
        vector<double> diff(v.size());
        transform(v.begin(), v.end(), diff.begin(), [mean
           ↪ ](double x) { return x - mean; });
        double sq_sum = inner_product(diff.begin(), diff.
           ↪ end(), diff.begin(), 0.0);
        double stdev = sqrt(sq_sum / v.size());
        return stdev;
}

ostream& operator<<(ostream& os, Event e) {
        os << "Event: " << e.eventtype << ", " << e.
           ↪ eventtime << " s";
        return os;
}
```

File: main.cc

```
#include "eventandstate.h"

#include <iostream>
#include <fstream>

using namespace std;

void stats(ofstream& ofs, vector<double> v) {
        double mean = calc_mean(v);
        double stddev = calc_stddev(v);
        double stddev_mean = stddev/sqrt(v.size());
        ofs << "mean = " << mean << endl;
        ofs << "stddev of mean estimate = " <<
           ↪ stddev_mean << endl;
        ofs << "95% conf int = " << mean - 1.96*
           ↪ stddev_mean << ", " << mean + 1.96*
           ↪ stddev_mean << endl;
        ofs << "90%␣conf␣int␣=␣" << mean − 1.645*
           ↪ stddev_mean << ",␣" << mean + 1.645*
           ↪ stddev_mean << endl;
        ofs << "99%␣conf␣int␣=␣" << mean − 2.576*
           ↪ stddev_mean << ",␣" << mean + 2.576*
           ↪ stddev_mean << endl;
        cout << endl;
}

int main() {
        cout << "Running␣task␣1␣" << endl;
        ofstream f_res("res_task3.txt");

        vector<double> ArrTime = {2., 1.5, 1.1};

        for(auto& dt : ArrTime) {
                EventList el;
                int seed = 0;
                default_random_engine rnd_engine(seed);


                //Kick−start:
                el.InsertEvent(Event::ArrivalQ1,
                   ↪ get_exp_time(rnd_engine, dt));
                el.InsertEvent(Event::Measure, 1000 +
                   ↪ get_exp_time(rnd_engine, 5));

                State s(rnd_engine, dt);

                while(s.nbr_measurements < 10000) {
                        cout << "LQ1␣=␣" << s.LQ1 << endl
                           ↪ ;
                        s.ProcessEvent(el);
                        auto it_temp = el.event_list.
                           ↪ erase(el.event_list.begin()
                           ↪ );
                        for(auto& e : el.event_list) cout
                           ↪ << e << endl;
                }

                s.Write("task3_"+to_string(dt)+".root");

                f_res << "Results␣for␣dtQ1␣=␣" << dt << "
                   ↪ ␣and␣LQ1␣" << endl;
                stats(f_res, s.v_LQ1);
```

```
                    f_res << "Results␣for␣dtQ1␣=␣" << dt << "
                        ↪ ␣and␣mean␣time␣" << endl;
                    stats(f_res, s.v_mean_time);
                    f_res << endl;
            }

            cout << "
                ↪ ########################################################
                ↪ " << endl;



}
```

7.4 Task 4

File: eventandstate.cc

```
#include "eventandstate.h"

#include <iostream>
#include <algorithm>
#include <math.h>

#include "TString.h"
#include "TFile.h"
#include "TGraph.h"
#include "TCanvas.h"
#include "TAxis.h"
#include "TColor.h"
#include "TLegend.h"
#include "TPaveText.h"
#include "TLatex.h"

using namespace std;

void EventList::InsertEvent(int type, double time) {
        Event e(type, time);
        if(event_list.size() == 0) {
                event_list.push_back(e);
                return;
        }

        auto it = event_list.begin();
        while(it != event_list.end() && e.eventtime > it
            ->eventtime) {
                ++it;
        }

        it = event_list.insert(it, e);
}

State::State(std::default_random_engine& re, unsigned int
   ↪  n_servers, double x_serve_time, double l, unsigned
   ↪  int measures, double m_time) : N(n_servers), x(
   ↪ x_serve_time), lambda(l), M(measures), T(m_time),
   ↪ nbr_measurements(0), rnd_engine(re) {
        NC = 0;
}

void State::ProcessEvent(EventList& el) {
        Event e = el.event_list[0];
        switch (e.eventtype){
                case Event::Arrival:
                        cout << "Arrival:␣" << e << endl;
                        if(NC >= N) {
                                el.InsertEvent(Event::
                                   ↪ Arrival, e.
                                   ↪ eventtime +
                                   ↪ get_exp_time(
                                   ↪ rnd_engine, 1./
                                   ↪ lambda));
                                break;
                        }
                        ++NC;
                        el.InsertEvent(Event::Arrival, e.
                           ↪ eventtime + get_exp_time(
                           ↪ rnd_engine, 1./lambda));
                        el.InsertEvent(Event::Depart, e.
                           ↪ eventtime + x);
                        break;
                case Event::Depart:
                        cout << "Depart:␣" << e << endl;
                        --NC;
                        break;
                case Event::Measure:
                        cout << "Measuring:␣" << NC << ",
                           ↪ ␣" << e.eventtime << endl;
                        ++nbr_measurements;
                        el.InsertEvent(Event::Measure, e.
                           ↪ eventtime + T);
                        v_NC.push_back(NC);
                        v_mean.push_back(calc_mean(v_NC))
                            ↪ ;
                        v_var.push_back(calc_stddev(v_NC)
                            ↪ );
                        v_time.push_back(e.eventtime);
                        break;
        }
}
void State::Write(string s) {
        cout << "Writing to file task4.root" << endl;
        TFile* f_out = new TFile(TString(s), "RECREATE");

        TGraph* g;
  TCanvas* C = new TCanvas();

  //double marker_size[runs] = {1, 2, 1, 2, 2};
  //double marker_style[runs] = {21, 21, 22, 22, 20};
  //red, green, blue, cyan, black
  //TColor marker_colour[runs] = {TColor(1,0,0), TColor
      ↪ (0,1,0), TColor(0,0,1), TColor(0,1,1), TColor
      ↪ (0,0,0)};
  TLegend* leg = new TLegend(0.7,0.7,0.55,0.9);
  //leg->SetHeader("Collimator","C"); // option "C"
      ↪ allows to center the header, WHY DON'T WORK?
  //leg->SetHeader("The Legend Title",""); // option "C"
      ↪ allows to center the header
  //TString legend_labels[runs] = {TString("Simple, 1 mm
      ↪ diameter"), TString("Simple, 1.5 mm diameter"),
      ↪ TString("Integrated cones (i)"),TString("
      ↪ Integrated cones (ii)"), TString("Integrated
      ↪ cylinders")};

        TString s_x[2] = {"Time (s)", "Time (s)"};
        TString s_y[4] = {"Number of customers", "Mean
            ↪ customers", "StdDev customers"};

        g = new TGraph(v_NC.size(), &(v_time[0]), &(v_NC
            ↪ [0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[0]);
        g->GetXaxis()->SetTitle(s_x[0]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_mean.size(), &(v_time[0]), &(
            ↪ v_mean[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[1]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_var.size(), &(v_time[0]), &(
            ↪ v_var[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[2]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();


        f_out->Close();


}

double get_exp_time(default_random_engine& rnd, double mu
    ↪ ) {
        exponential_distribution<double> dist_exp(1/mu);
        double rand = dist_exp(rnd);
        cout << "rnd␣=␣" << rand << endl;
        return rand;
}

double calc_mean(vector<double>& v) {
        double sum = accumulate(v.begin(), v.end(), 0.0);
        return sum/v.size();
}

double calc_stddev(vector<double>& v) {
        double mean = calc_mean(v);
        vector<double> diff(v.size());
        transform(v.begin(), v.end(), diff.begin(), [mean
            ↪ ](double x) { return x - mean; });
        double sq_sum = inner_product(diff.begin(), diff.
            ↪ end(), diff.begin(), 0.0);
        double stdev = sqrt(sq_sum / v.size());
        return stdev;
}

ostream& operator<<(ostream& os, Event e) {
        os << "Event:␣" << e.eventtype << ",␣" << e.
            ↪ eventtime << "␣s";
        return os;
}
```

File: main.cc

```
#include "eventandstate.h"
#include "write.h"

#include <iostream>
#include <fstream>

using namespace std;

int main() {
        cout << "Running␣task␣1␣" << endl;
        ofstream f_res("res_task3.txt");

        vector<double> N = {1000, 1000, 1000, 100, 100,
            ↪ 100, 100, 10000};
        vector<double> x = {100, 10, 200, 10, 10, 10, 10,
            ↪  10};
        vector<double> lambda = {8, 80, 4, 4, 4, 4, 4,
            ↪ 100};
```

```
vector<double> M = {1000, 1000, 1000, 1000, 4000,
    ↪ 4000, 10000, 1000};
vector<double> T = {1, 1, 1, 4, 1, 4, 0.1, 1};
vector<double> M_delay = {0, 0, 0, 10, 10, 10,
    ↪ 10, 0};

for(unsigned int j = 0; j != x.size(); ++j) {
        EventList el;
        int seed = 0;
        default_random_engine rnd_engine(seed);

        //Kick-start:
        el.InsertEvent(Event::Arrival,
            ↪ get_exp_time(rnd_engine, 1./lambda[
            ↪ j]));
        el.InsertEvent(Event::Measure, M_delay[j]
            ↪ + T[j]);

        State s(rnd_engine, N[j], x[j], lambda[j
            ↪ ], M[j], T[j]);

        while(s.nbr_measurements < M[j]) {
                s.ProcessEvent(el);
                auto it_temp = el.event_list.
                    ↪ erase(el.event_list.begin()
                    ↪ );
                //for(auto& e : el.event_list)
                    ↪ cout << e << endl;
        }

        s.Write("task4_"+to_string(j)+".root");

        write_txt(s.v_NC, "task4_"+to_string(j)+"
            ↪ .txt");

        f_res << "Results␣for␣run␣=␣" << j <<
            ↪ endl;
        f_res << "mean(NC)␣=␣" << calc_mean(s.
            ↪ v_NC) << endl;
        f_res << "stddev(NC)␣=␣" << calc_stddev(s
            ↪ .v_NC) << endl;
        f_res << "nbr_measurements␣=␣" << s.
            ↪ nbr_measurements << endl;
        f_res << endl;
}
```

```
        cout << "
            ↪ ################################################
            ↪ " << endl;



}
```

7.5 Task 5

File: procs.cc

```
#include "procs.h"

#include <iostream>
#include <algorithm>

using namespace std;


Signal::Signal(int type, string dest, double time, int
    ↪ meas) : SignalType(type), ArrivalTime(time),
    ↪ Destination(dest), Meas(meas) { }

ostream& operator<<(ostream& os, Signal s) {
        os << "signal:␣" << s.SignalType << ",␣" << s.
            ↪ Destination << ",␣" << s.ArrivalTime;
        return os;
}

void Process::AddSignal(int type, string dest, double
    ↪ time, int meas) {
        Signal s(type, dest, time, meas);
        if(SignalList.size() == 0) {
                SignalList.push_back(s);
                return;
        }
        auto it = SignalList.begin();
        while(it != SignalList.end() && s.ArrivalTime >=
            ↪ it->ArrivalTime) {
                ++it;
        }

        it = SignalList.insert(it, s);
}
```

```
void Process::RemoveSignal() {
        if(SignalList.size() != 0) {
                //delete SignalList[0];
                SignalList.erase(SignalList.begin());
        }
        else {
                std::cout << "SignalList empty!!!!" <<
                    ↪ std::endl;
                exit(0);
        }
}
void ProcessList::AddProcess(shared_ptr<Process> p) {
        procs.push_back(p);
}

void ProcessList::Update() {
        //cout << "Updating process list" << endl;
        //Sorting the process list
        sort(procs.begin(), procs.end(),
                        [](const shared_ptr<Process> a,
                            ↪ const shared_ptr<Process> b
                            ↪ ) {
                        if(a->SignalList.size() == 0)
                            ↪ return false;
                        else if(b->SignalList.size() ==
                            ↪ 0) return true;
                        else if(a->SignalList.size() == 0
                            ↪  && b->SignalList.size() ==
                            ↪  0) return false;
                        else return a->SignalList[0].
                            ↪ ArrivalTime < b->SignalList
                            ↪ [0].ArrivalTime;
                        }
                        );
}

void ProcessList::TreatSignal() {
        //cout << "Treating signal from ProcessList" <<
            ↪ endl;
        Signal x = procs[0]->SignalList[0];
        shared_ptr<Process> p = FetchProcess(x);
        //cout << "About to invoke TreatSignal!" << endl;
        if(!p) cout << "No process with name " << x.
            ↪ Destination << endl;
        p->TreatSignal(x);
        procs[0]->RemoveSignal();
```

```
}

shared_ptr<Process> ProcessList::FetchProcess(Signal x) {
        //cout << "Fetching process with name: " << x.
            ↪ Destination << endl;
        auto it = find_if(procs.begin(), procs.end(),
                        [x] (const shared_ptr<Process> p)
                            ↪  {
                                return x.Destination == p
                                    ↪ ->GetName();
                                }
                        );
        return (*it);
}

std::ostream& operator<<(std::ostream& os, ProcessList pl
    ↪ ) {
        os << "Listing␣processes:␣" << endl;
        for(auto p : pl.procs) {
                if(p->SignalList.size() == 0) continue;
                os << p->GetName() << "␣-␣";
                for(auto& s : p->SignalList) {
                        os << s << "␣";
                }
                os << endl;
        }
        return os;
}

std::ostream& operator<<(ostream& os, Process* p) {
        for(unsigned int i = 0; i != p->SignalList.size()
            ↪ ; ++i) {
                os << i << "␣" << p->SignalList[i] <<
                    ↪ endl;
        }
        return os;
}

void Generator::TreatSignal(Signal x) {
        //cout << "Treating signal in " << this->GetName
            ↪ () << " - " << x << endl;
        switch(x.SignalType) {
                case Signal::Ready:
                        ++nbr_arrivals;
                        AddSignal(Signal::Arrival, "Q" +
                            ↪ to_string(this->Load->GetQ(
                            ↪ rnd_engine)), x.ArrivalTime
                            ↪ );
                        //AddSignal(Signal::Arrival, "Q"
                            ↪ + to_string(1), x.
                            ↪ ArrivalTime);
                        AddSignal(Signal::Ready, "
                            ↪ Generator", x.ArrivalTime +
                            ↪  get_uni_time(rnd_engine,
                            ↪ t_mean));
                        //for(auto& s : SignalList) cout
                            ↪ << s << endl;
                        break;
        }
}

void Queue::TreatSignal(Signal x) {
        //cout << "Treating signal in " << this->GetName
            ↪ () << " - " << x << endl;
        double t_mean = 0.5;
        switch(x.SignalType) {
                case Signal::Ready:
                        if(LQ > 1){
                                //cout << "Queue larger
                                    ↪ than 1 and adding
                                    ↪ Ready" << endl;
                                AddSignal(Signal::Ready,
                                    ↪ this->GetName(), x.
                                    ↪ ArrivalTime +
                                    ↪ get_exp_time(
                                    ↪ rnd_engine, t_mean)
                                    ↪ );
                                ++nbr_ready;
                        }
                        --LQ;
                        //for(auto& s : SignalList) cout
                            ↪ << s << endl;
                        break;
                case Signal::Arrival:
                        if(LQ == 0) {
                                AddSignal(Signal::Ready,
                                    ↪ this->GetName(), x.
                                    ↪ ArrivalTime +
                                    ↪ get_exp_time(
                                    ↪ rnd_engine, t_mean)
                                    ↪ );
                                //cout << "Queue was
```

```
                                        ↪ empty and adding
                                        ↪ Ready" << endl;
                    }
                    ++LQ;
                    break;
            case Signal::Measure:
                    AddSignal(Signal::Arrival, "
                        ↪ Measure", x.ArrivalTime, LQ
                        ↪ );
                    break;
        }
}
void Measure::TreatSignal(Signal x) {
        //cout << "Treating signal in " << this->GetName
            ↪ () << " - " << x << endl;
        switch(x.SignalType) {
            case Signal::Ready:
                    AddSignal(Signal::Ready, this->
                        ↪ GetName(), x.ArrivalTime +
                        ↪ get_exp_time(rnd_engine,
                        ↪ t_mean));
                    for(int j = 1; j != 6; ++j) {
                            AddSignal(Signal::Measure
                                ↪ , "Q"+to_string(j),
                                ↪  x.ArrivalTime);
                    }
                    break;
            case Signal::Arrival:
                    if(measured < 4) {
                            LQ += x.Meas;
                            ++measured;
                    }
                    else {
                            LQ += x.Meas;
                            ++measured;
                            ++nbr_measurements;
                            //cout << "Measuring: "
                                ↪ << LQ << endl;
                            v_LQ.push_back(LQ/1);
                            v_time.push_back(x.
                                ↪ ArrivalTime);
                            measured = LQ = 0;
                    }
                    break;
        }
}
```

```
#ifdef ROOTSYS
#include "TString.h"
#include "TFile.h"
#include "TGraph.h"
#include "TCanvas.h"
#include "TAxis.h"
#include "TColor.h"
#include "TLegend.h"
#include "TPaveText.h"
#include "TLatex.h"

void Measure::Write(string s) {
        cout << "Writing␣to␣file␣task4.root" << endl;
        TFile* f_out = new TFile(TString(s), "RECREATE");

        TGraph* g;
  TCanvas* C = new TCanvas();

  //double marker_size[runs] = {1, 2, 1, 2, 2};
  //double marker_style[runs] = {21, 21, 22, 22, 20};
  //red, green, blue, cyan, black
  //TColor marker_colour[runs] = {TColor(1,0,0), TColor
     ↪ (0,1,0), TColor(0,0,1), TColor(0,1,1), TColor
     ↪ (0,0,0)};
  TLegend* leg = new TLegend(0.7,0.7,0.55,0.9);
  //leg->SetHeader("Collimator","C"); // option "C"
     ↪ allows to center the header, WHY DON'T WORK?
  //leg->SetHeader("The Legend Title",""); // option "C"
     ↪ allows to center the header
  //TString legend_labels[runs] = {TString("Simple, 1 mm
     ↪ diameter"), TString("Simple, 1.5 mm diameter"),
     ↪ TString("Integrated cones (i)"),TString("
     ↪ Integrated cones (ii)"), TString("Integrated
     ↪ cylinders")};

        TString s_x[2] = {"Time␣(s)", "Time␣(s)"};
        TString s_y[4] = {"Number␣of␣customers", "Mean␣
            ↪ customers", "StdDev␣customers"};

        g = new TGraph(v_LQ.size(), &(v_time[0]), &(v_LQ
            ↪ [0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
```

```
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[0]);
        g->GetXaxis()->SetTitle(s_x[0]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_mean.size(), &(v_time[0]), &(
            ↪ v_mean[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪ draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[1]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_var.size(), &(v_time[0]), &(
            ↪ v_var[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪ draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[2]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();


        f_out->Close();


}
#endif
```

```
unsigned int RandomLoad::GetQ(std::default_random_engine&
    ↪  rnd) {
        //cout << "Random Load " << endl;
        uniform_int_distribution<int> dist_uni(1, 5);
        int rand = dist_uni(rnd);
        //cout << "rnd = " << rand << endl;
        return rand;
}

unsigned int RobinLoad::GetQ(std::default_random_engine&
    ↪ rnd) {
        //cout << "Robin Load " << endl;
        if(index > 5) index = 1;
        return index++;
}

unsigned int SmallestQueLoad::GetQ(std::
    ↪ default_random_engine& rnd) {
        unsigned int small = Qs[0]->LQ;
        vector<unsigned int> v_small = {1};
        for(unsigned int j = 1; j != Qs.size(); ++j) {
                if(small == Qs[j]->LQ) v_small.push_back(
                    ↪ j+1);
                else if(small > Qs[j]->LQ) {
                        small = Qs[j]->LQ;
                        v_small.clear();
                        v_small = {j+1};
                }
        }
        if(v_small.size() == 1) return v_small[0];
        //for(auto& t : v_small) cout << t << ", ";
        //cout << "v_small.size() = " << v_small.size()
            ↪ << endl;
        uniform_int_distribution<unsigned int> dist_uni
            ↪ (0, v_small.size()-1);
        unsigned int rand = dist_uni(rnd);
        //cout << "Rand = " << rand << endl;
        //if(v_small[rand] == 0) cout << "SmallestQueLoad
            ↪  = 0" << endl;
        return v_small[rand];

}

double get_exp_time(default_random_engine& rnd, double mu
    ↪ ) {
        exponential_distribution<double> dist_exp(1/mu);
        double rand = dist_exp(rnd);
        return rand;
}

double get_uni_time(default_random_engine& rnd, double mu
   ↪ ) {
        uniform_real_distribution<double> dist_uni(0, 2*
           ↪ mu);
        double rand = dist_uni(rnd);
        //cout << "rnd = " << rand << endl;
        return rand;
}

double calc_mean(vector<double>& v) {
        double sum = accumulate(v.begin(), v.end(), 0.0);
        return sum/v.size();
}

double calc_stddev(vector<double>& v) {
        double mean = calc_mean(v);
        vector<double> diff(v.size());
        transform(v.begin(), v.end(), diff.begin(), [mean
           ↪ ](double x) { return x - mean; });
        double sq_sum = inner_product(diff.begin(), diff.
           ↪ end(), diff.begin(), 0.0);
        double stdev = sqrt(sq_sum / v.size());
        return stdev;
}
```

File: main.cc

```
#include "procs.h"
#include "write.h"

#include <fstream>

using namespace std;

void stats(ofstream& ofs, vector<double> v) {
        double mean = calc_mean(v);
        double stddev = calc_stddev(v);
        double stddev_mean = stddev/sqrt(v.size());
        ofs << "mean␣=␣" << mean << endl;
        ofs << "stddev␣of␣mean␣estimate␣=␣" <<
           ↪ stddev_mean << endl;
        ofs << "95% conf int = " << mean - 1.96*
            ↪ stddev_mean << ", " << mean + 1.96*
            ↪ stddev_mean << endl;
        ofs << "90% conf int = " << mean - 1.645*
            ↪ stddev_mean << ", " << mean + 1.645*
            ↪ stddev_mean << endl;
        ofs << "99% conf int = " << mean - 2.576*
            ↪ stddev_mean << ", " << mean + 2.576*
            ↪ stddev_mean << endl;
        cout << endl;
}

int main() {
        //cout/gc << "Running task 5" << endl;
        ofstream f_res("res_task5.txt");

        vector<double> v_t = {0.11, 0.15, 2.};
        //vector<double> v_t = {0.30};
        for(auto& m_time : v_t) {
                for(int i = 0; i!=3;++i) {
                        int seed = 1;
                        default_random_engine rnd(seed);

                        double mean_arrival_time = m_time
                            ↪ ;
                        double mean_measure_time = 5;
                        vector<shared_ptr<Queue>> Qs;
                        ProcessList plist;

                        for(int j = 1; j != 6; ++j) {
                                //cout/gc << "Creating
                                    ↪ process: " << "Q" +
                                    ↪  to_string(j) <<
                                    ↪ endl;
                                Qs.push_back(shared_ptr<
                                    ↪ Queue>(new Queue("Q
                                    ↪ "+to_string(j), rnd
                                    ↪ )));
                                plist.AddProcess(Qs[j-1])
                                    ↪ ;
                        }

                        shared_ptr<Measure> M(new Measure
                            ↪ ("Measure", rnd,
                            ↪ mean_measure_time));
                        plist.AddProcess(M);
```

```
shared_ptr<LoadDistr> rnd_load(
    ↪ new RandomLoad(Qs));
shared_ptr<LoadDistr> robin_load(
    ↪ new RobinLoad(Qs));
shared_ptr<LoadDistr> opt_load(
    ↪ new SmallestQueLoad(Qs));
vector<shared_ptr<LoadDistr>>
    ↪ loads = {rnd_load,
    ↪ robin_load, opt_load};

shared_ptr<Generator> G(new
    ↪ Generator("Generator", rnd,
    ↪  mean_arrival_time, loads[i
    ↪ ]));
plist.AddProcess(G);

G->AddSignal(Signal::Ready, "
    ↪ Generator", get_uni_time(
    ↪ rnd, mean_arrival_time));
M->AddSignal(Signal::Ready, "
    ↪ Measure", get_exp_time(rnd,
    ↪  mean_measure_time));

plist.Update();
//cout/gc << plist << endl;

int j = 0;
while(plist.procs[0]->SignalList
    ↪ [0].ArrivalTime < 100000) {
//while(M->nbr_measurements < 2)
    ↪ {
          plist.TreatSignal();
          //cout << "Before sorting
              ↪  ... " << endl;
          //cout << plist << endl;
          plist.Update();
          //cout << plist;
          //cout << "LQ = " << Qs
              ↪ [0]->LQ << endl;
          int sum = 0;
          for(int i = 0; i!=5; ++i)
              ↪  sum += Qs[i]->
              ↪ nbr_ready;
          //cout/gc << "nbr_ready =
              ↪  " << sum << endl;
```

```
                                    //cout/gc << "
                                        ↪ nbr_arrivals = " <<
                                        ↪  G->nbr_arrivals <<
                                        ↪  endl;
                                    ++j;
                                    //cout << ">>>>>>>>>> "
                                        ↪ << endl;
                              }

#ifdef ROOTSYS
                              M->Write("task5_"+to_string(
                                  ↪ mean_arrival_time)+loads[i
                                  ↪ ]->GetName()+".root");
#endif

                              write_txt(M->v_LQ, "task5_"+
                                  ↪ to_string(mean_arrival_time
                                  ↪ )+loads[i]->GetName()+".txt
                                  ↪ ");

                              f_res << "Run with mean arrival
                                  ↪ time " << mean_arrival_time
                                  ↪  << " and Load distr " <<
                                  ↪ loads[i]->GetName() << endl
                                  ↪ ;
                              stats(f_res, M->v_LQ);
                              f_res << "Little's theorem LQ = "
                                  ↪  << (1./mean_arrival_time)
                                  ↪ *0.5 << endl;
                              f_res << endl;
                        }
                  }


}
```

7.6 Task 6

File: eventandstate.cc

```
#include "eventandstate.h"

#include <iostream>
#include <algorithm>
#include <math.h>
```

```
#include "TString.h"
#include "TFile.h"
#include "TGraph.h"
#include "TCanvas.h"
#include "TAxis.h"
#include "TColor.h"
#include "TLegend.h"
#include "TPaveText.h"
#include "TLatex.h"

using namespace std;

void EventList::InsertEvent(int type, double time) {
        Event e(type, time);
        if(event_list.size() == 0) {
                event_list.push_back(e);
                return;
        }

        auto it = event_list.begin();
        while(it != event_list.end() && e.eventtime > it
            ↪ ->eventtime) {
                ++it;
        }

        it = event_list.insert(it, e);
}

State::State(std::default_random_engine& re, double l) :
    ↪ lambda(l), nbr_measurements(0), rnd_engine(re) {
        NC = 0;
}

void State::ProcessEvent(EventList& el) {
        Event e = el.event_list[0];
        double new_time;
        switch (e.eventtype){
                case Event::Arrival:
                        cout << "Arrival:␣" << e << endl;
                        ++NC;
                        new_time = e.eventtime +
                            ↪ get_uni_time(rnd_engine);
                        if(new_time < 8*60) {
                                el.InsertEvent(Event::
                                    ↪ Arrival, new_time);
                        }
                        if(NC == 1) {
                                el.InsertEvent(Event::
                                   ↪ Depart, e.eventtime
                                   ↪  + get_uni_time(
                                   ↪ rnd_engine));
                        }
                        v_arrival_time.push_back(e.
                           ↪ eventtime);
                        break;
                case Event::Depart:
                        cout << "Depart: " << e << endl;
                        --NC;
                        if(NC > 0) el.InsertEvent(Event::
                           ↪ Depart, e.eventtime +
                           ↪ get_uni_time(rnd_engine));
                        v_mean_time.push_back(e.eventtime
                           ↪  - v_arrival_time[0]);
                        v_arrival_time.erase(
                           ↪ v_arrival_time.begin());
                        break;
                        /*
                case Event::Measure:
                        cout << "Measuring: " << NC << ",
                           ↪   " << e.eventtime << endl;
                        ++nbr_measurements;
                        el.InsertEvent(Event::Measure, e.
                           ↪ eventtime + T);
                        v_NC.push_back(NC);
                        v_mean.push_back(calc_mean(v_NC))
                           ↪ ;
                        v_var.push_back(calc_stddev(v_NC)
                           ↪ );
                        v_time.push_back(e.eventtime);
                        break;
                        */
        }
}
void State::Write(string s) {
        cout << "Writing to file task4.root" << endl;
        TFile* f_out = new TFile(TString(s), "RECREATE");

        TGraph* g;
  TCanvas* C = new TCanvas();

  //double marker_size[runs] = {1, 2, 1, 2, 2};
  //double marker_style[runs] = {21, 21, 22, 22, 20};
```

```
//red, green, blue, cyan, black
//TColor marker_colour[runs] = {TColor(1,0,0), TColor
    ↪ (0,1,0), TColor(0,0,1), TColor(0,1,1), TColor
    ↪ (0,0,0)};
TLegend* leg = new TLegend(0.7,0.7,0.55,0.9);
//leg->SetHeader("Collimator","C"); // option "C"
    ↪ allows to center the header, WHY DON'T WORK?
//leg->SetHeader("The Legend Title",""); // option "C"
    ↪ allows to center the header
//TString legend_labels[runs] = {TString("Simple, 1 mm
    ↪ diameter"), TString("Simple, 1.5 mm diameter"),
    ↪ TString("Integrated cones (i)"),TString("
    ↪ Integrated cones (ii)"), TString("Integrated
    ↪ cylinders")};

        TString s_x[2] = {"Time␣(s)", "Time␣(s)"};
        TString s_y[4] = {"Number␣of␣customers", "Mean␣
            ↪ customers", "StdDev␣customers"};

        g = new TGraph(v_NC.size(), &(v_time[0]), &(v_NC
            ↪ [0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[0]);
        g->GetXaxis()->SetTitle(s_x[0]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_mean.size(), &(v_time[0]), &(
            ↪ v_mean[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[1]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();

        g = new TGraph(v_var.size(), &(v_time[0]), &(
            ↪ v_var[0]));
        g->Draw("ACP"); //For the first one, one needs to
            ↪  draw axis with "A". Option "SAME" is not
            ↪ needed with TGraph!
        //Axis objects for TGraph are created after it
            ↪ has been drawn, thus they need to be
            ↪ defined here.
        g->SetTitle("");
        g->GetYaxis()->SetTitle(s_y[2]);
        g->GetXaxis()->SetTitle(s_x[1]);
        g->GetYaxis()->CenterTitle();
        g->GetXaxis()->CenterTitle();
        g->Write();


        f_out->Close();


}

double get_exp_time(default_random_engine& rnd, double mu
    ↪ ) {
        exponential_distribution<double> dist_exp(1/mu);
        double rand = dist_exp(rnd);
        cout << "rnd␣=␣" << rand << endl;
        return rand;
}

double calc_mean(vector<double>& v) {
        double sum = accumulate(v.begin(), v.end(), 0.0);
        return sum/v.size();
}

double calc_stddev(vector<double>& v) {
        double mean = calc_mean(v);
        vector<double> diff(v.size());
        transform(v.begin(), v.end(), diff.begin(), [mean
            ↪ ](double x) { return x - mean; });
        double sq_sum = inner_product(diff.begin(), diff.
            ↪ end(), diff.begin(), 0.0);
        double stdev = sqrt(sq_sum / v.size());
        return stdev;
```

```
}

ostream& operator<<(ostream& os, Event e) {
        os << "Event:␣" << e.eventtype << ",␣" << e.
           ↪ eventtime << "␣s";
        return os;
}

double get_uni_time(default_random_engine& rnd) {
        uniform_real_distribution<double> dist_uni(10,
           ↪ 20);
        double rand = dist_uni(rnd);
        cout << "rnd␣=␣" << rand << endl;
        return rand;
}
```

File: main.cc

```
#include "eventandstate.h"
#include "write.h"

#include <iostream>
#include <fstream>

using namespace std;

void stats(ofstream& ofs, vector<double> v) {
        double mean = calc_mean(v);
        double stddev = calc_stddev(v);
        double stddev_mean = stddev/sqrt(v.size());
        ofs << "mean␣=␣" << mean << endl;
        ofs << "stddev␣of␣mean␣estimate␣=␣" <<
           ↪ stddev_mean << endl;
        ofs << "95%␣conf␣int␣=␣" << mean - 1.96*
           ↪ stddev_mean << ",␣" << mean + 1.96*
           ↪ stddev_mean << endl;
        ofs << "90%␣conf␣int␣=␣" << mean - 1.645*
           ↪ stddev_mean << ",␣" << mean + 1.645*
           ↪ stddev_mean << endl;
        ofs << "99%␣conf␣int␣=␣" << mean - 2.576*
           ↪ stddev_mean << ",␣" << mean + 2.576*
           ↪ stddev_mean << endl;
        cout << endl;
}

int main() {
```

```
cout << "Running␣task␣1␣" << endl;
ofstream f_res("res_task6.txt");

double lambda = 4./60.;
unsigned int days = 1000;

vector<double> overtime, mean_time;

int seed = 0;
default_random_engine rnd_engine(seed);

State s(rnd_engine, lambda);

for(unsigned int j = 0; j != days; ++j) {
        cout << "
            ↪ **********************************
            ↪ " << endl;
        cout << "Day␣" << j << endl;

        EventList el;

        //Kick-start:
        el.InsertEvent(Event::Arrival,
            ↪ get_exp_time(rnd_engine, 1./lambda)
            ↪ );

        double last_time;
        while(el.event_list.size() > 0) {
                last_time = el.event_list[0].
                    ↪ eventtime;
                s.ProcessEvent(el);
                auto it_temp = el.event_list.
                    ↪ erase(el.event_list.begin()
                    ↪ );
                cout << "LQ␣=␣" << s.NC << endl;
                //cout << "Event list: " << endl;
                //for(auto& e : el.event_list)
                    ↪ cout << e << endl;
        }

        overtime.push_back(last_time);

}

//s.Write("task6.root");
```

```
        f_res << "Results␣for␣run␣" << endl;
        stats(f_res, overtime);
        stats(f_res, s.v_mean_time);
        f_res << endl;
        cout << "
            ↪ #########################################
            ↪ " << endl;



}
```

7.7 Task 7

File: main.cc

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <random>
#include <algorithm>

using namespace std;

struct Component {
        int nbr;
        double t_end;
};

double get_uni_time(default_random_engine& rnd) {
        uniform_real_distribution<double> dist_uni(1, 5);
        double rand = dist_uni(rnd);
        return rand;
}

void remove_comp(vector<Component>& sys, int nbr) {
        auto it = find_if(sys.begin(), sys.end(),
                        [nbr](const Component& c) {
                                return c.nbr == nbr;
                                }
                        );
        if(it == sys.end()) return;
        sys.erase(it);
}
```

```
ostream& operator<<(ostream& os, Component& c) {
        os << "Component:␣" << c.nbr << ",␣" << c.t_end;
        return os;
}

double calc_mean(vector<double>& v) {
        double sum = accumulate(v.begin(), v.end(), 0.0);
        return sum/v.size();
}

double calc_stddev(vector<double>& v) {
        double mean = calc_mean(v);
        vector<double> diff(v.size());
        transform(v.begin(), v.end(), diff.begin(), [mean
            ↪ ](double x) { return x - mean; });
        double sq_sum = inner_product(diff.begin(), diff.
            ↪ end(), diff.begin(), 0.0);
        double stdev = sqrt(sq_sum / v.size());
        return stdev;
}

void stats(ofstream& ofs, vector<double> v) {
        double mean = calc_mean(v);
        double stddev = calc_stddev(v);
        double stddev_mean = stddev/sqrt(v.size());
        ofs << "mean␣=␣" << mean << endl;
        ofs << "stddev␣of␣mean␣estimate␣=␣" <<
            ↪ stddev_mean << endl;
        ofs << "95%␣conf␣int␣=␣" << mean - 1.96*
            ↪ stddev_mean << ",␣" << mean + 1.96*
            ↪ stddev_mean << endl;
        ofs << "90%␣conf␣int␣=␣" << mean - 1.645*
            ↪ stddev_mean << ",␣" << mean + 1.645*
            ↪ stddev_mean << endl;
        ofs << "99%␣conf␣int␣=␣" << mean - 2.576*
            ↪ stddev_mean << ",␣" << mean + 2.576*
            ↪ stddev_mean << endl;
        cout << endl;
}

int main() {
        cout << "Running␣task␣7␣" << endl;
        ofstream f_res("res_task7.txt");

        vector<Component> sys;
```

```
vector<double> v_life;

int seed = 0;
default_random_engine rnd(seed);

int runs = 1000;

for(int i = 0; i != runs; ++i) {
        for(int j = 1; j != 6; ++j) {
                sys.push_back({j, get_uni_time(
                   ↪ rnd)});
        }

        sort(sys.begin(),sys.end(),
                        [](const Component& a,
                           ↪ const Component& b)
                           ↪  {
                        return a.t_end < b.t_end;
                        }
                        );

        for(auto& c : sys) cout << c << endl;

        double last_time;
        while(sys.size() != 0) {
                cout << "␣  ␣" << endl;
                Component c = sys[0];
                last_time = c.t_end;
                remove_comp(sys, c.nbr);
                if(c.nbr == 1) {
                        remove_comp(sys, 2);
                        remove_comp(sys, 5);
                }
                else if(c.nbr == 3) {
                        remove_comp(sys, 3);
                        remove_comp(sys, 4);
                }
        for(auto& c : sys) cout << c << endl;
        }

        cout << "Life␣time␣=␣" << last_time <<
            ↪ endl;
        v_life.push_back(last_time);

}
```

```
        //s.Write("task6.root");

        f_res << "Results for run " << endl;
        stats(f_res, v_life);
        f_res << endl;



}
```