Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

**Лабораторная работа №3**

**Дисциплина «Многопоточное программирование»**

**Выполнил:**

Съестов Дмитрий Вячеславович

Группа P3417

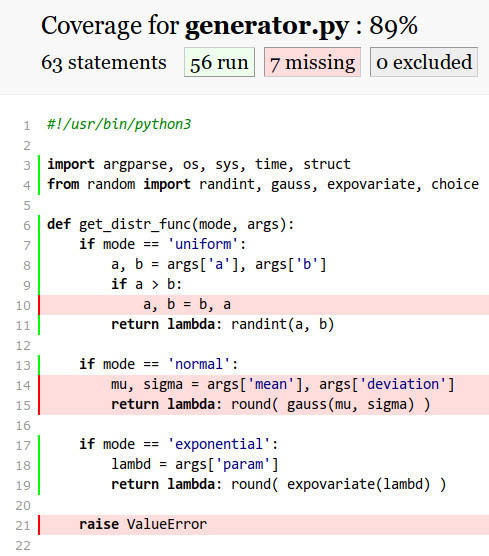
**Преподаватель:**

Дергачёв Андрей Михайлович

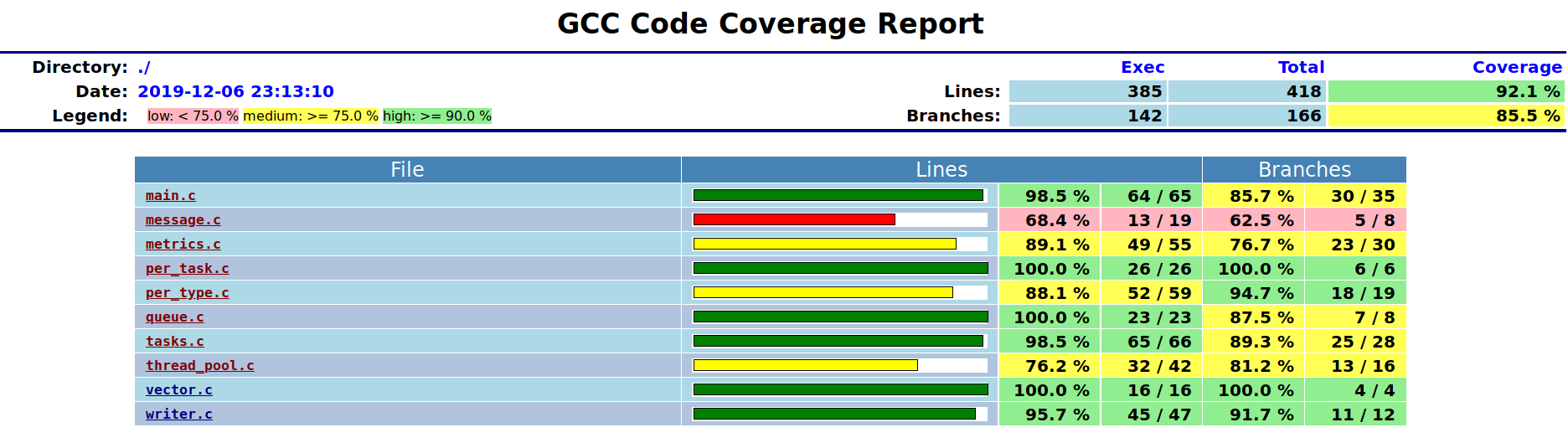
Санкт-Петербург

2019

|  |
| --- |
| **Задание**  1. Разработать обработчик задач POSIX.  2. Разработать генератор задач, поддерживающий несколько распределений.  3. Обеспечить необходимое тестовое покрытие и снять метрики производительности. |

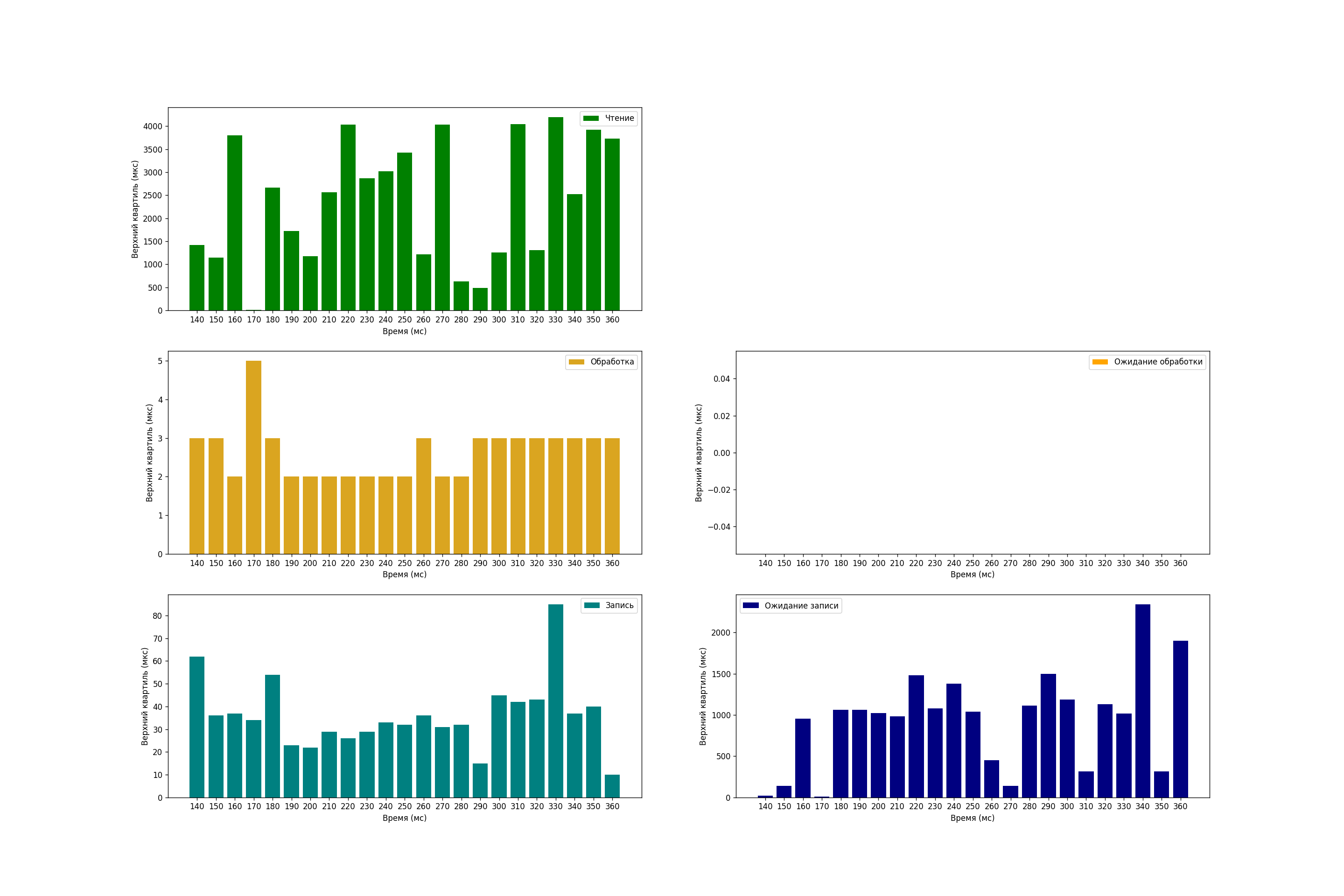
Тестовое покрытие генератора: 89% (приведён скриншот HTML-отчёта, сгенерированный пакетом pytest-cov, используя данные о покрытии, собранные coverage.py)

Тестовое покрытие обработчика: 92.1% (приведён скриншот HTML-отчёта, сгенерированный Python-пакетом Gcovr, используя данные о покрытии, собранные gcov)



Метрики производительности снимались следующим образом: каждые n миллисекунд вычислялся верхний квартиль (т. е. 75-й перцентиль) накопленных за это время метрик, после чего эти данные выводились в файл. Затем файл с метриками обрабатывался скриптом на Python, генерирующим графики.

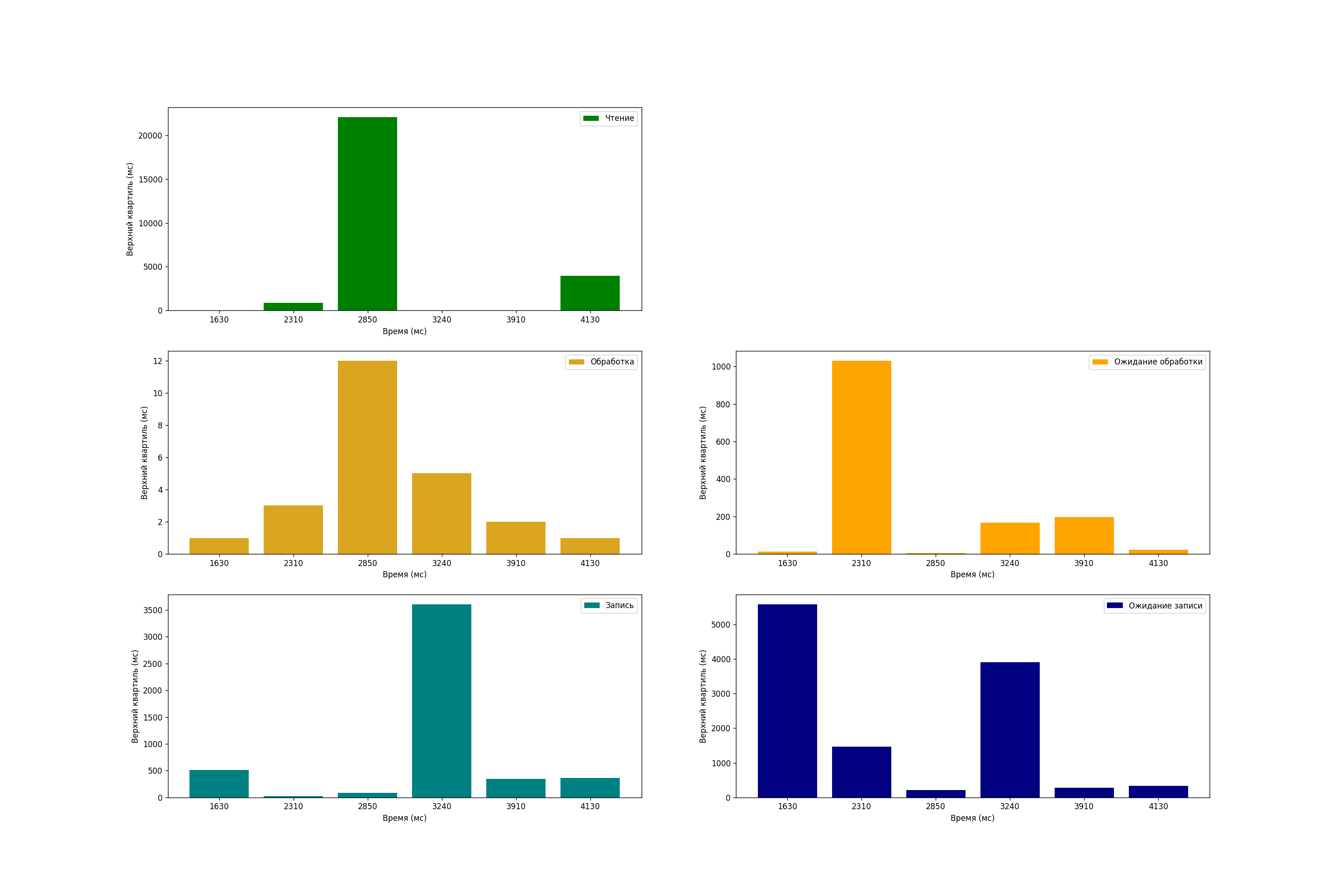
$ ./generator.py 100 exponential -p 0.5 | ./processor -n 10



Экспоненциально распределённые значения близки к нулю, благодаря чему достигается низкий интервал ожидания в генераторе, а следовательно, относительно быстрое чтение.

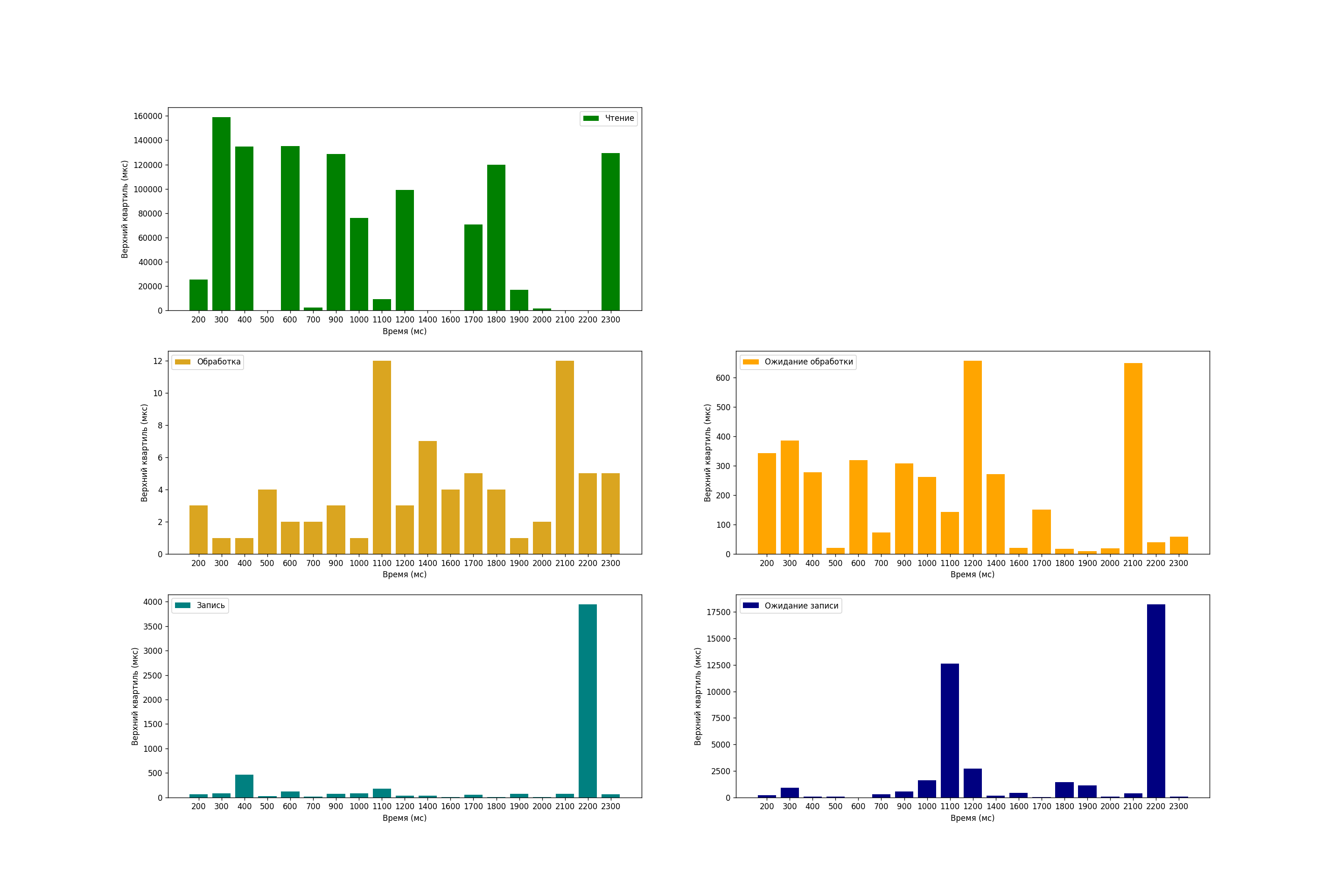
По умолчанию используется стратегия PER\_TASK, в которой на каждую задачу создаётся поток. Поэтому время ожидания равно нулю.

$ ./generator.py 100 normal -m 50 -d 25 | ./processor -s per\_type -n 10

Нормально распределённые значения имеют матожидание 50, что гораздо больше, чем в предыдущем случае. Поэтому чтение происходит значительно медленнее, чем при экспоненциальном распределении.

Также, поскольку каждый поток специализируется на одном типе задач, поток в определённый момент времени может простаивать, в то время как другой поток загружен.

$ ./generator.py 100 uniform -a -100 -b 100 | ./processor -s thread\_pool -t 10 -n 100

Здесь мы видим, что время чтения медленнее, чем даже при нормальном распределении. Проблема заключается в том, что половина значений оказалась отрицательными, что вызывало исключения в генераторе при попытке выждать межзадачный интервал. Генератор написан на Python, и обработка исключений значительно замедлила его работу.

Время ожидания оказалось распределено равномернее, чем при стратегии PER\_TYPE, за счёт отсутствия специализации потоков и большего их количества.

Ещё одна интересная особенность заключается в том, что в конце работы возникла огромная очередь на запись. Это связано с тем, что на обработку поступило несколько длинных массивов, которые выводятся в файл дольше всего, и в то же время чтение проходило быстро, загружая систему работой.

**Выводы**

Время чтения зависит от распределения. У экспоненциального оно ниже всего, а выше всего — при распределениях, часто выдающих отрицательные числа, из-за возникновения исключений в генераторе.

Время обработки стабильно низкое, т. к. обработка не использует системные вызовы.

Время ожидания обработки равно нулю при PER\_TASK (нет очередей), относительно равномерно при THREAD\_POOL и колеблется при PER\_TYPE.

Время записи может колебаться из-за массивов и высокой загрузки системы.