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)

Coverage for **generator.py** : 89%

63 statements | 56 run | 7 missing | 0 excluded

```
1  #!/usr/bin/python3
2
3  import argparse, os, sys, time, struct
4  from random import randint, gauss, expovariate, choice
5
6  def get_distr_func(mode, args):
7      if mode == 'uniform':
8          a, b = args['a'], args['b']
9          if a > b:
10             a, b = b, a
11         return lambda: randint(a, b)
12
13     if mode == 'normal':
14         mu, sigma = args['mean'], args['deviation']
15         return lambda: round( gauss(mu, sigma) )
16
17     if mode == 'exponential':
18         lambd = args['param']
19         return lambda: round( expovariate(lambd) )
20
21     raise ValueError
22
```

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

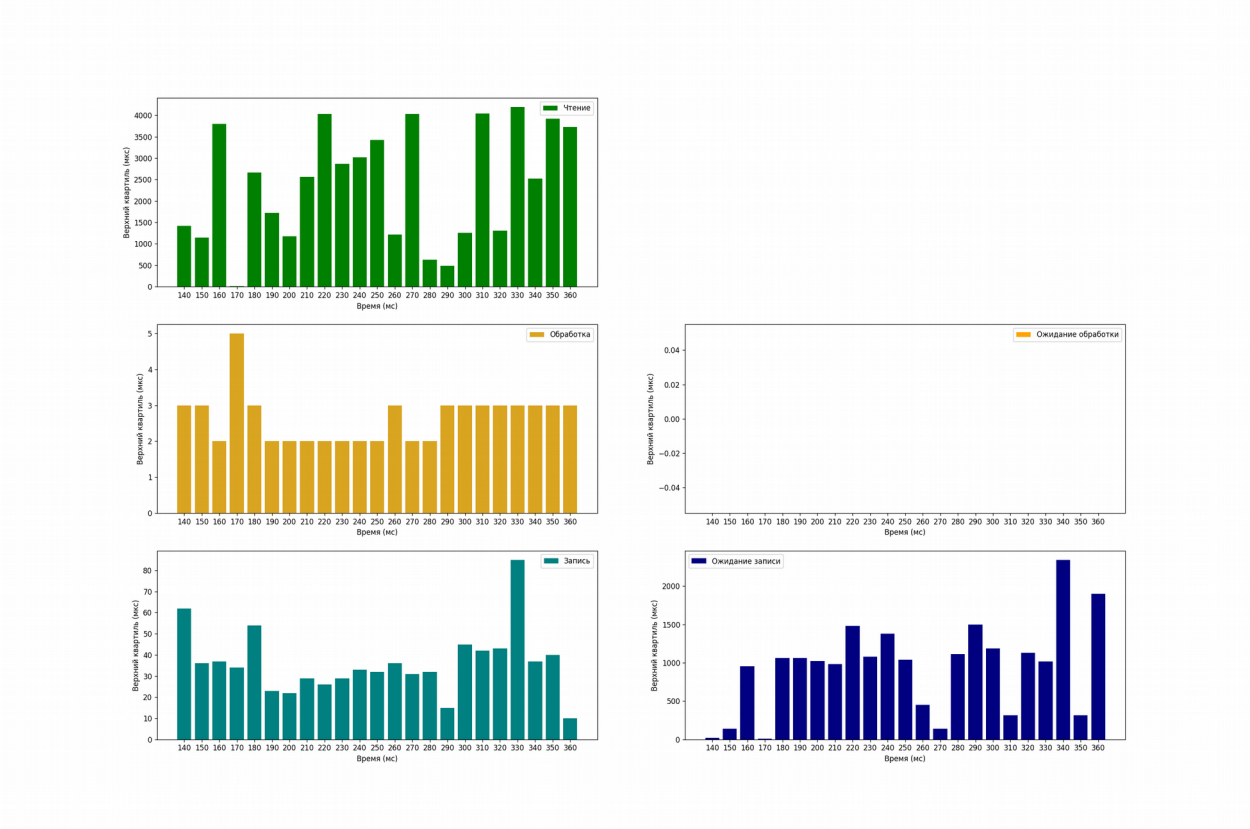
GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./	Lines:	385	418	92.1 %
Date:	2019-12-06 23:13:10	Branches:	142	166	85.5 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %				

File	Lines			Branches	
main.c		98.5 %	64 / 65	85.7 %	30 / 35
message.c		68.4 %	13 / 19	62.5 %	5 / 8
metrics.c		89.1 %	49 / 55	76.7 %	23 / 30
per_task.c		100.0 %	26 / 26	100.0 %	6 / 6
per_type.c		88.1 %	52 / 59	94.7 %	18 / 19
queue.c		100.0 %	23 / 23	87.5 %	7 / 8
tasks.c		98.5 %	65 / 66	89.3 %	25 / 28
thread_pool.c		76.2 %	32 / 42	81.2 %	13 / 16
vector.c		100.0 %	16 / 16	100.0 %	4 / 4
writer.c		95.7 %	45 / 47	91.7 %	11 / 12

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.

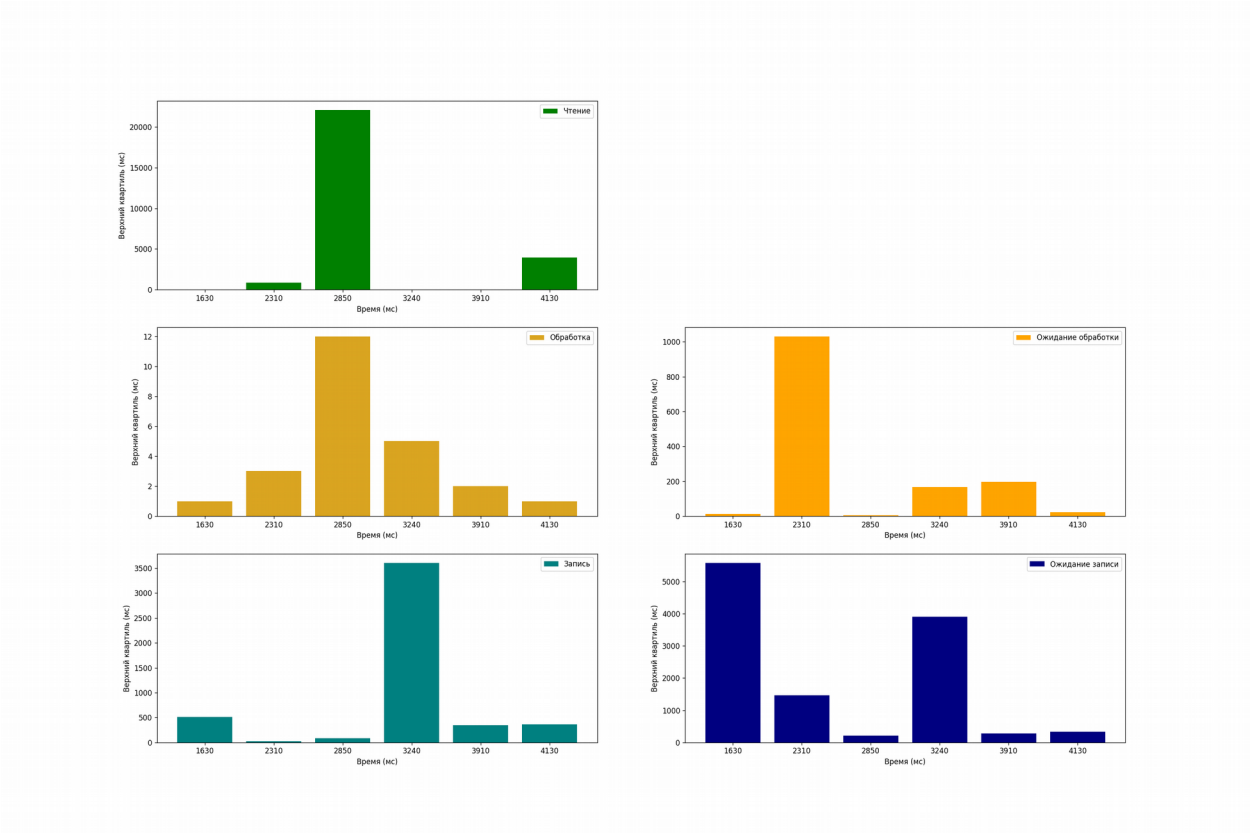
```
$  ./generator.py 100 exponential -p 0.5 | ./processor -n 10
```



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.

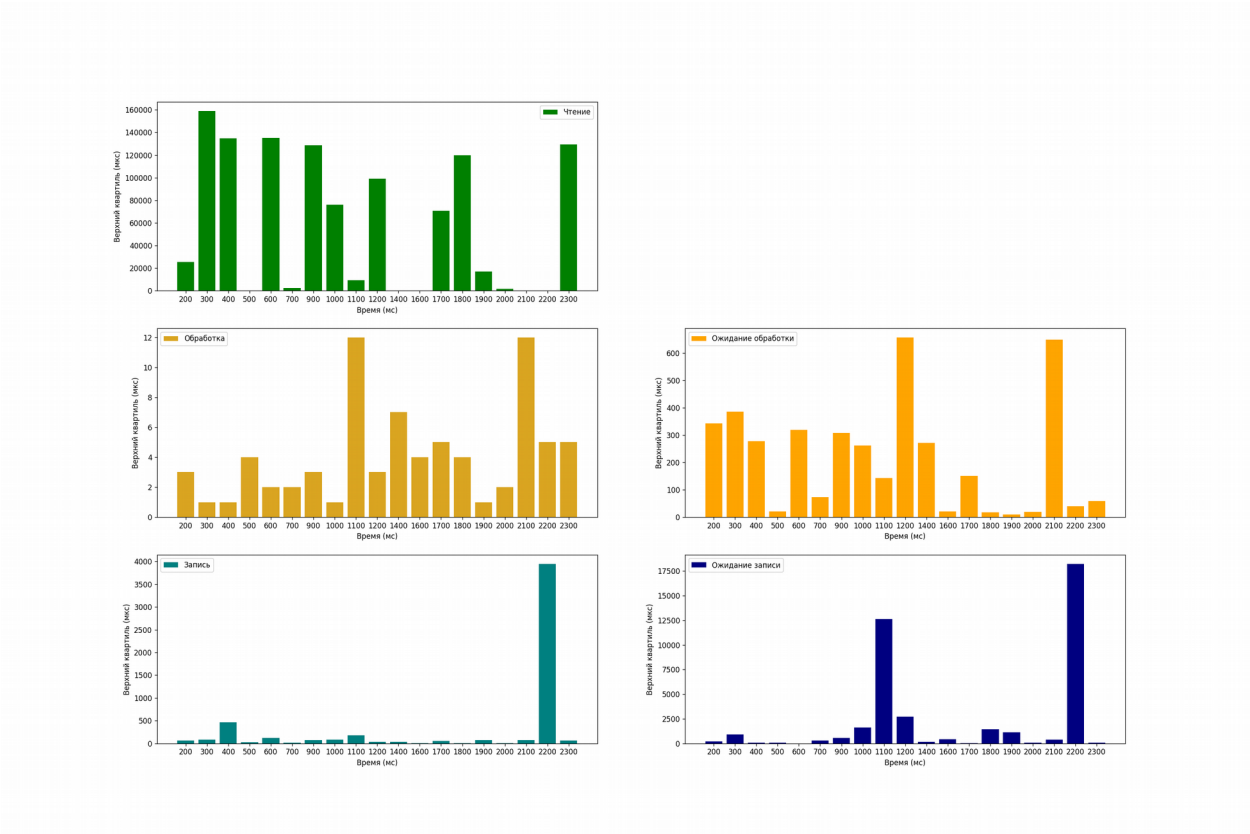
```
$  ./generator.py 100 normal -m 50 -d 25 | ./processor -s per_type -n 10
```



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.