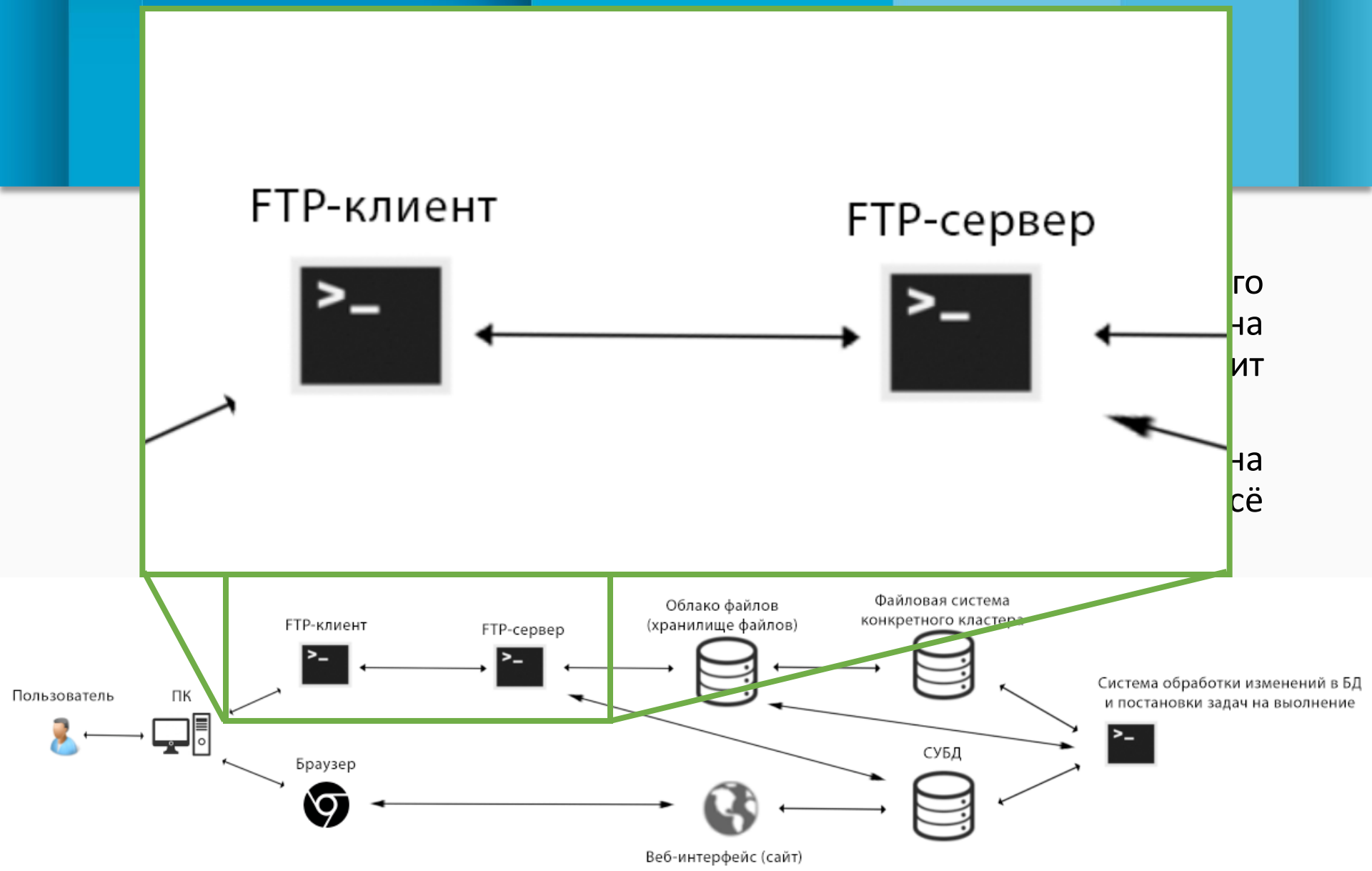
Разработка метода разделения потока данных для передачи файлов от пользователя на суперкомпьютер с учетом приоритетов

Студент: Осадчук Дмитрий Русланович

Научный руководитель: Сальников Алексей Николаевич



Упрощённая структура Uniclust

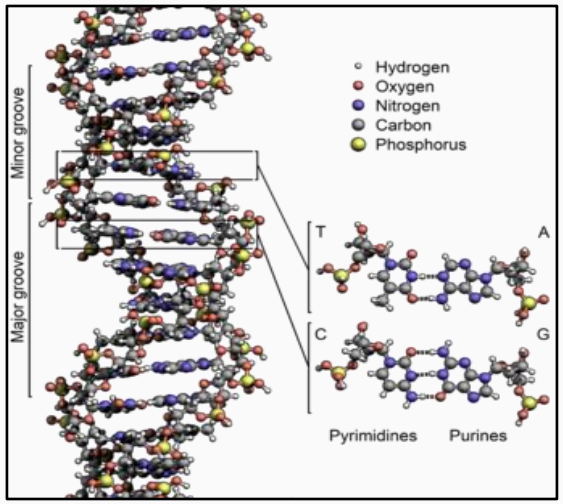
Файлы большого размера

Проблематика

Для многих задач требуется обрабатывать файлы большого размера. Их размер может достигать нескольких Терабайт.

К таким задачам относятся задачи моделирования динамических систем, биоинформатические задачи, задачи выделения требуемых характеристик из больших объёмов данных для машинного обучения и другие.

Анализ аэродинамических характеристик самолёта



Сборка генома

Что такое Uniclust и зачем он нужен?

Необходимость в дозагрузке и системе приоритетов

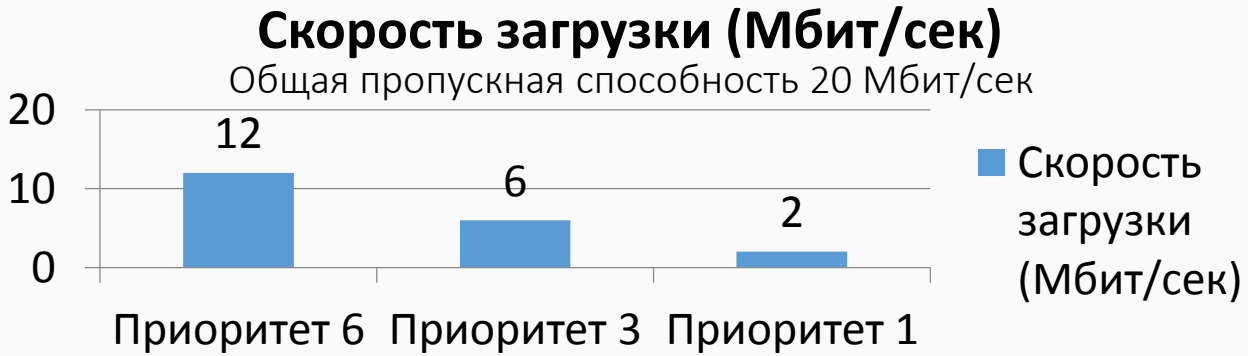
Необходимость в дозагрузке и системе приоритетов

Проблематика

Передача файлов такого размера почти всегда превышает время сессии работы абонента с суперкомпьютером, поэтому возникает необходимость в дозагрузке файлов при повторном включении.

Кроме того, для удобного и точного планирования задач возникает необходимость в загрузке каких-то файлов быстрее, чем других. Это возможно при последовательной передаче, однако это не даёт «гибко» планировать задачи на суперкомпьютере.

Было принято решение передавать файлы параллельно и ввести систему приоритетов. То есть задавать загрузкам приоритет, в зависимости от которого, какие-то файлы бы загружались быстрее или медленнее, чем другие.



Файлы большого размера
Список требований

Цель и задачи

Проблематика

Целью данной работы является разработка клиент-серверного программного обеспечения, которое позволит пользователю осуществлять двунаправленный параллельный обмен файлами с учётом приоритетов между своим ПК (клиентом) и файловым облаком (сервером).

Задачи:

1) Разработать алгоритм разделения потока данных в соответствии с заданными приоритетами, который эффективно использует предоставленный канал связи;
2) Реализовать возможность «дозагрузки» файлов при повторном сеансе;
3) Предоставить пользователю удобный графический интерфейс;
4) Разработанное ПО должно быть кроссплатформенным;

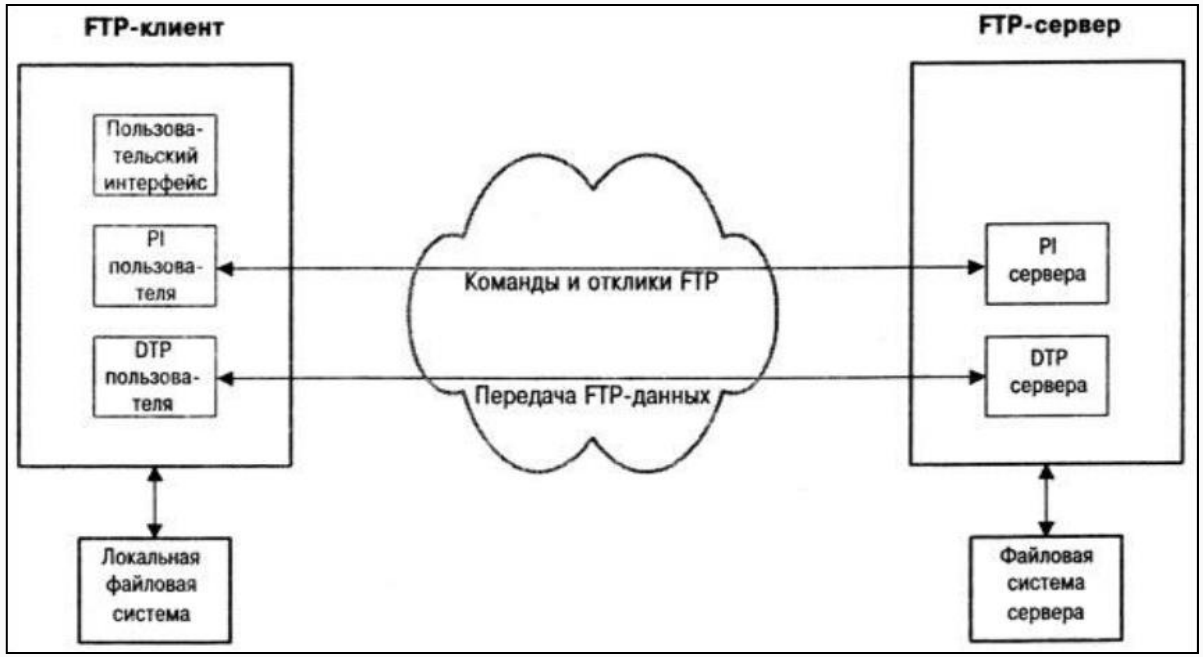
Необходимость в дозагрузке и системе приоритетов

FTP-протокол

FTP-протокол

Проблематика

Было принято решение использовать протокол **FTP** (File Tranfer Protocol).
1 FTP-подключение = 1 «командное» + 1 «информационное» соединение.



Для приёма/передачи каждого из файлов используется одно FTP-подключение.

Разработанные алгоритмы

Реализация

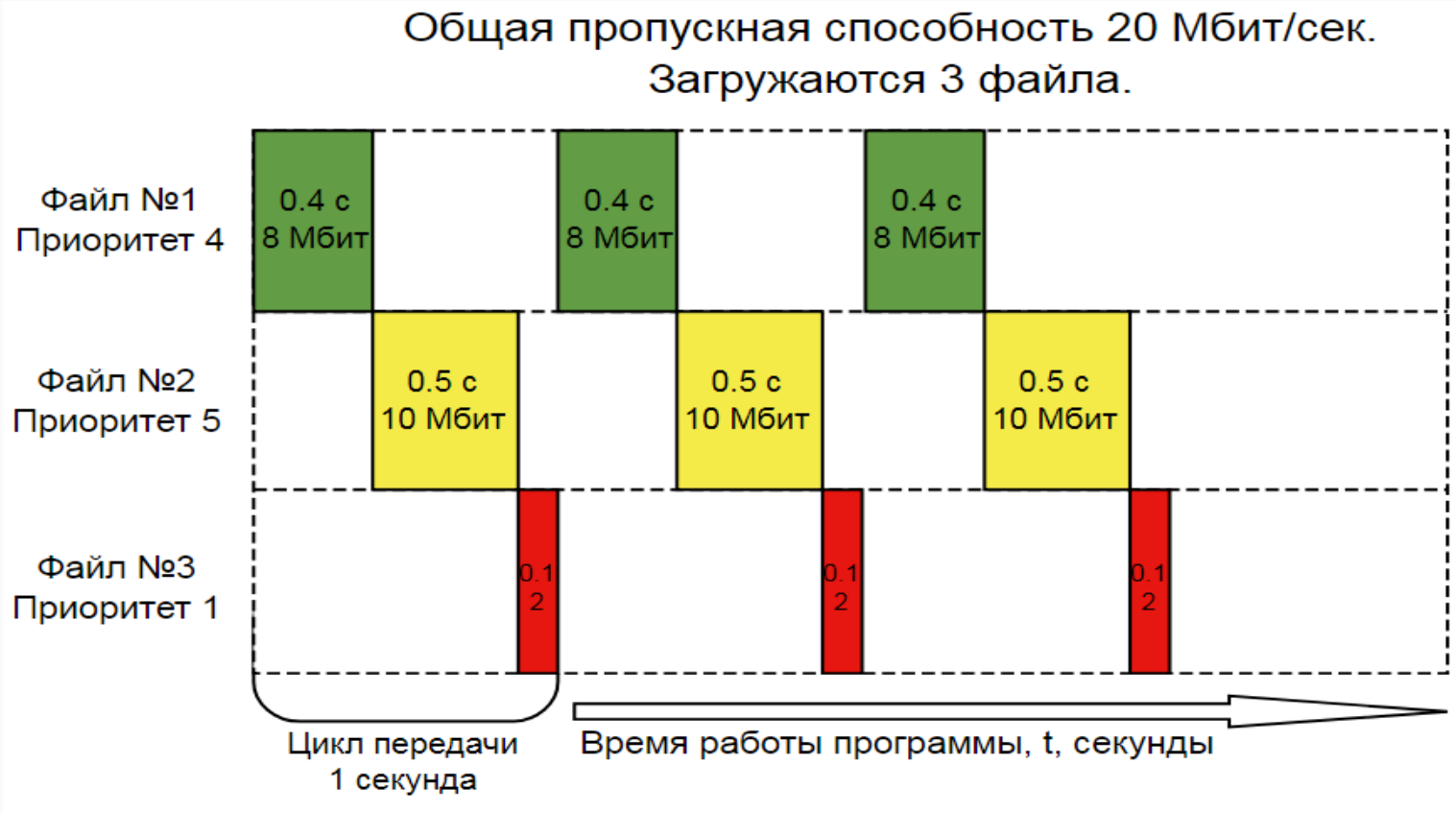
В ходе работы было разработано два алгоритма разделения потока данных:

1. «по времени передачи» – алгоритм основан на алгоритмах планирования выполнения параллельных процессов в операционных системах;
2. «по кадрам передачи» - доработанный алгоритм, в котором «цикл передачи» измеряется не во времени, а в количестве данных.

Метод «по времени передачи»

Реализация

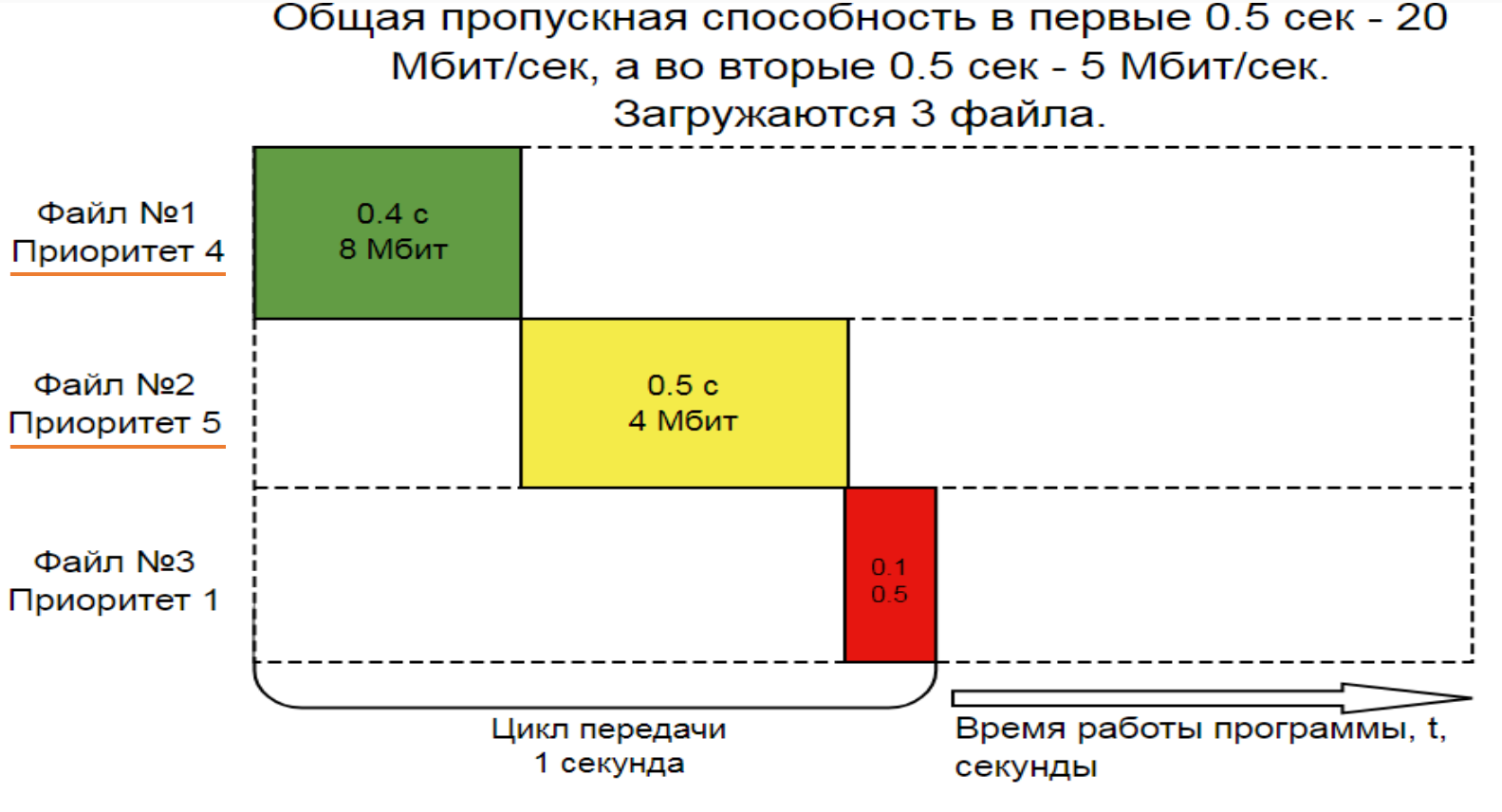
Каждый файл передаётся определённое количество времени, которое зависит от приоритета, с максимальной скоростью, и так далее очень много раз.



Проблема «приоритетной несправедливости» метода разделения потока данных «по времени передачи»

Реализация

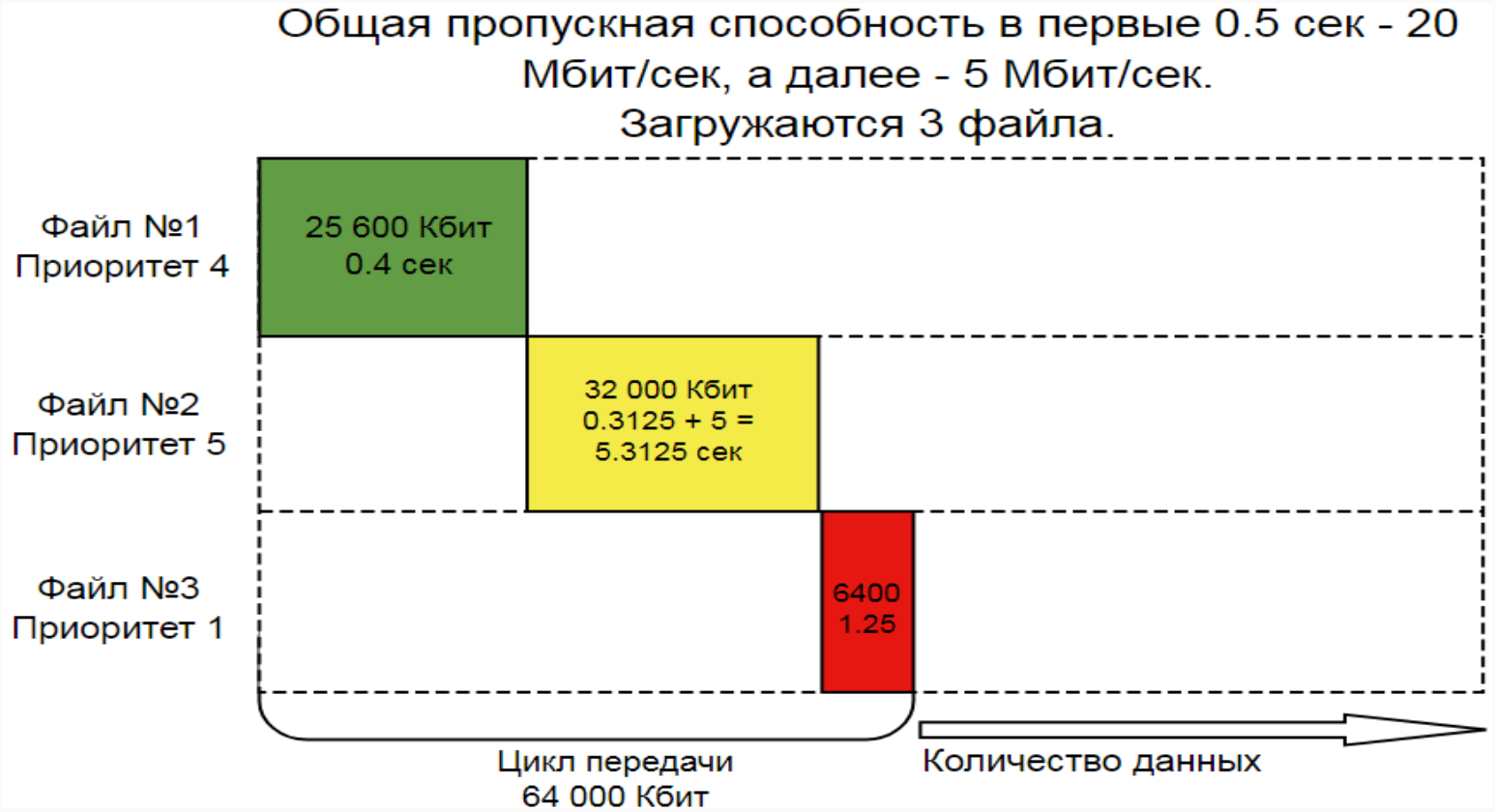
При переменной пропускной способности данный метод «несправедливо» разделяет поток данных относительно заданных приоритетов.



Метод «по кадрам передачи»

Реализация

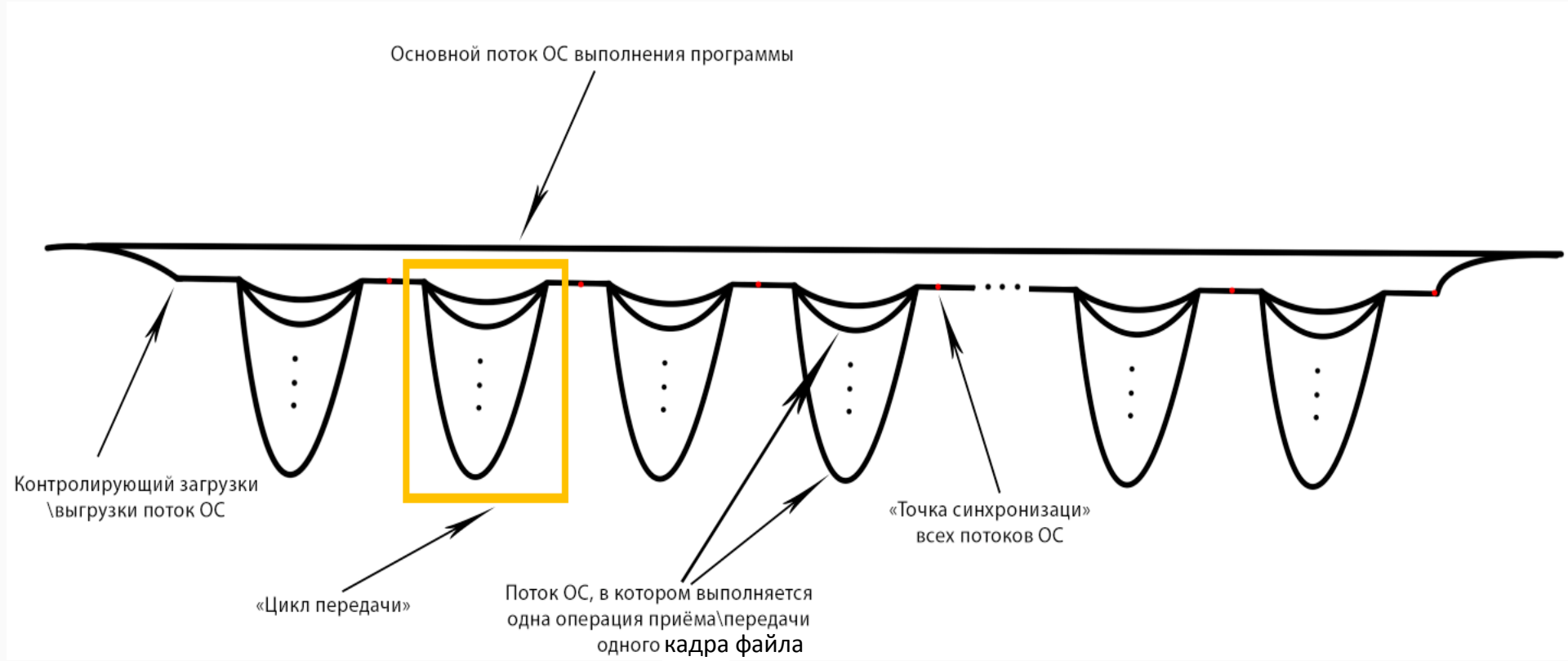
Цикл передачи теперь будем измерять не во времени, а в количестве передаваемых данных. Каждая часть данных, переданная за один цикл передачи – «кадр файла».



Структура потоков ОС при работе программы

Реализация

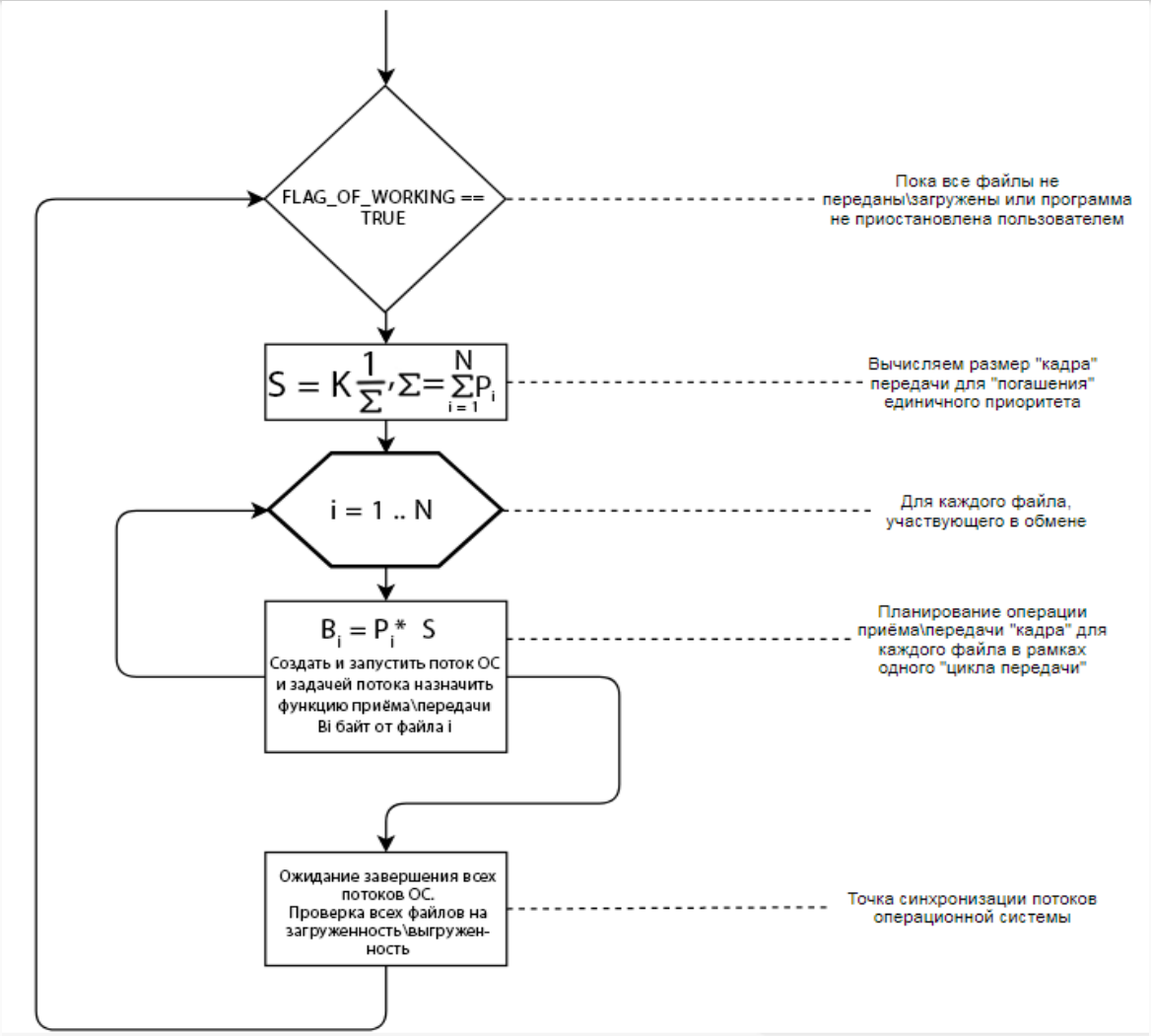
Для достижения лучшего эффекта «параллелизма» было принято решение в рамках одного цикла передачи использовать отдельные потоки для передачи каждого из кадров файлов.



Метод «по кадрам передачи»
Блок-схема разработанного алгоритма

Блок-схема разработанного алгоритма

Реализация

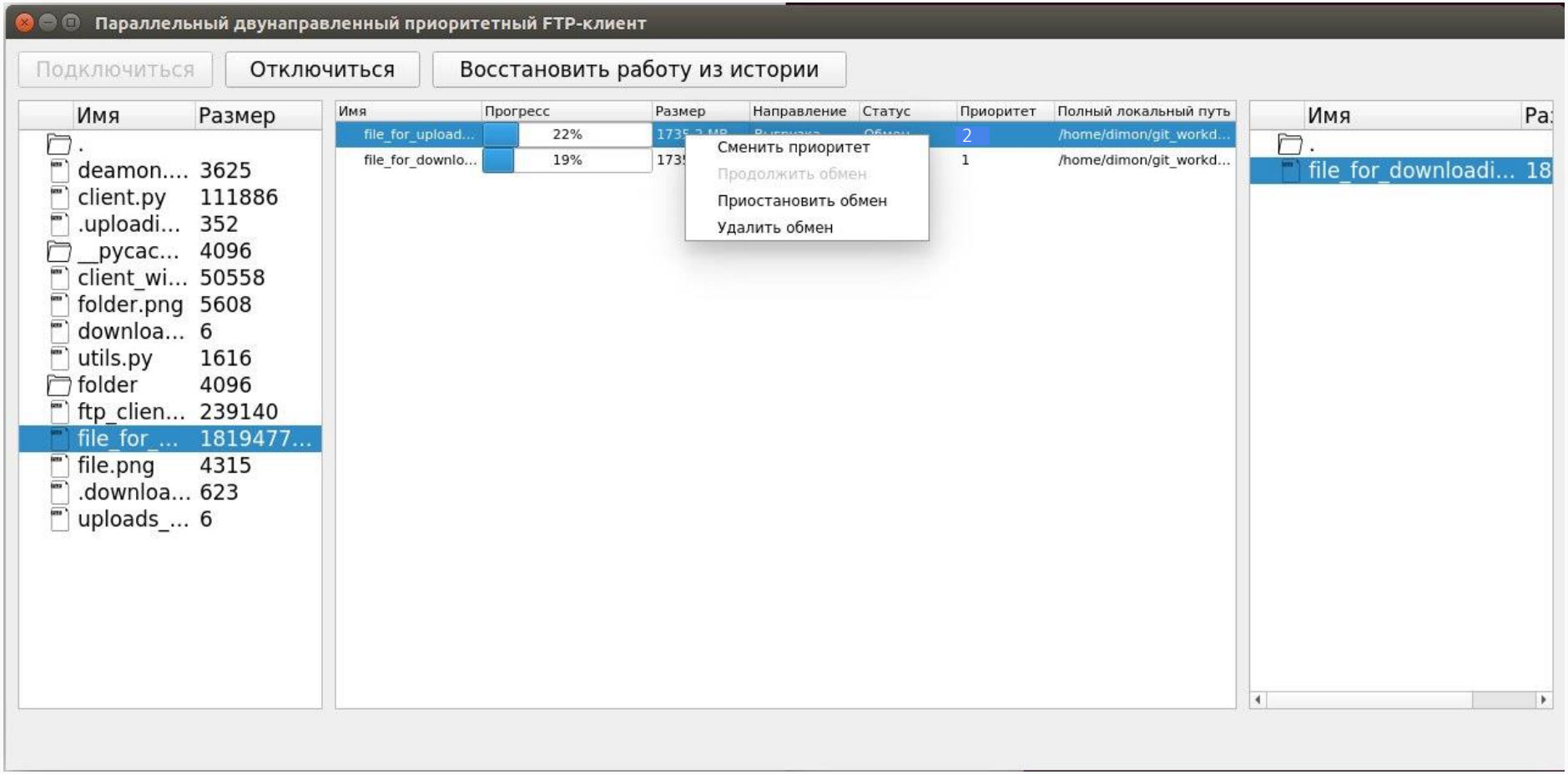


Структура ОС при работе программы

Клиентская часть приложения (FTP-клиент)

Клиентская часть приложения (FTP-клиент)

Разработанное программное обеспечение



Написано около 2300 строк кода на Python 3 (функциональная + графическая часть)

Блок-схема разработанного алгоритма

Серверная часть приложения (FTP-сервер)

Серверная часть приложения (FTP-сервер)

Разработанное программное обеспечение

Серверная часть приложения была разработана с использованием асинхронной модели обработки соединений с клиентами. То есть, для обработки каждого соединения не создаётся отдельный поток, как в многопоточной модели, а используются функции обратного вызова с мультиплексированным механизмом потоков ввода/вывода. Это позволяет не блокировать работу сервера при приёме (записи в файл) или передаче (чтении из файла) файла, а продолжать её и принимать или передавать параллельно сразу множетсво файлов.

Для разработки использовался язык Python 3 и модуль pyftpdlib. Благодаря модулю реализация серверной части приложения заняла около 230 строк кода.

Клиентская часть приложения (FTP-клиент)

Блок-схема разработанного алгоритма

Методика тестирования

Тестирование и результаты

Тестирование проводилось на факультете ВМК МГУ. Использовались 14 компьютеров с установленными разными пропускными способностями каналов связи в качестве клиентов и 1 выделенный кластер, на котором был запущен сервер.

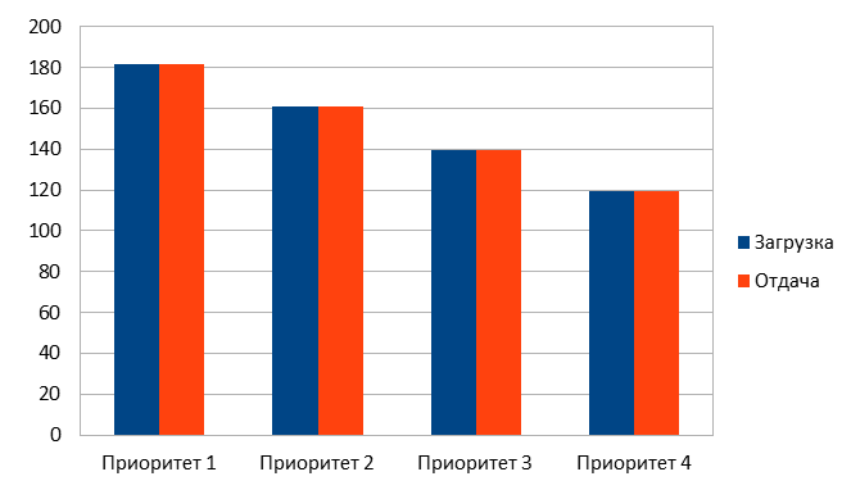
Был произведён двунаправленный параллельный обмен файлами разного размера (по 16, 32 и 64 Мбайт) с установленными различными приоритетами.

Полученные результаты совпали с ожидаемыми. Метод разделения потока данных «по времени передачи» распределял поток некорректно при переменной пропускной способности канала связи, в то время как метод «по кадрам передачи» отрабатывал корректно как при постоянной пропускной способности, так и при переменном поведении этой величины.

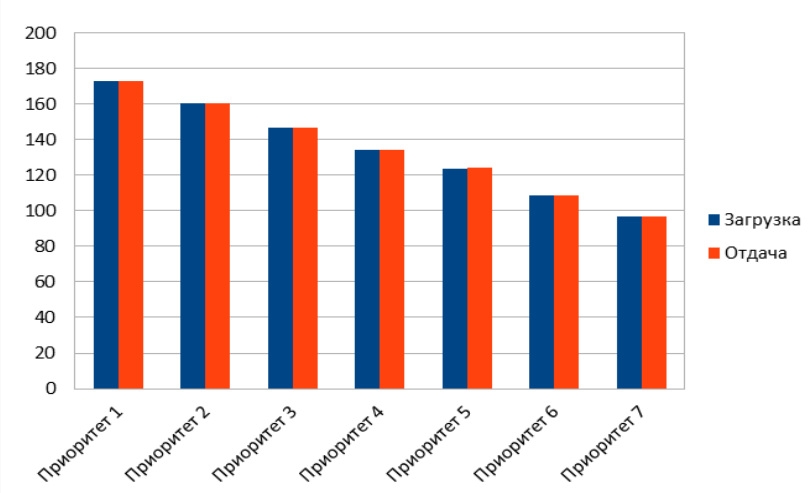
Метод «по кадрам передачи» Блок-схема разработанного алгоритма

Тест №1

Приём N файлов и передача N файлов одинакового размера с приоритетом K, где N – порядковый номер клиента, а K – порядковый номер файла.



Клиент №4

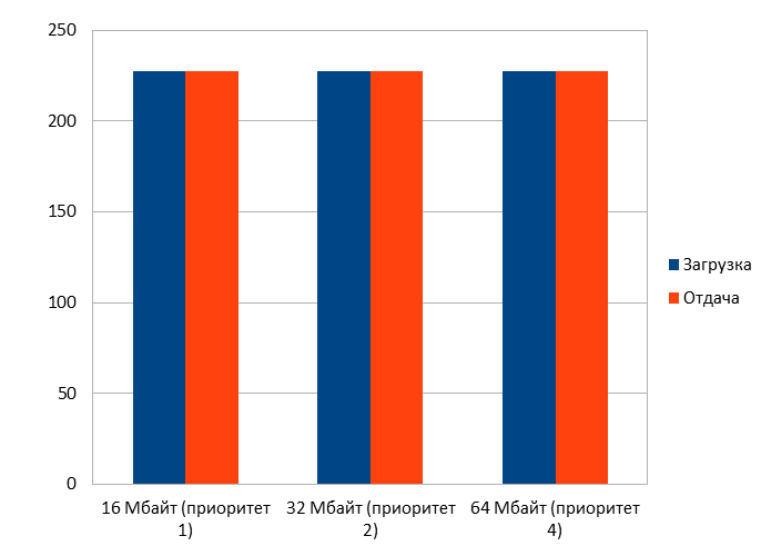


Клиент №7

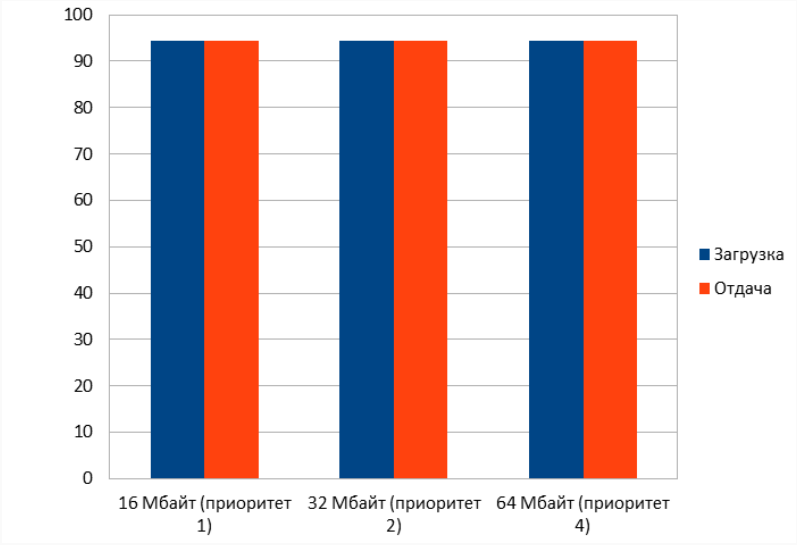
Вывод: файлы одинакового размера с разным приоритетом передались за разное время (чем больше приоритет, тем меньше время загрузки\передачи).

Тест №2

Приём 3 файлов и передача 3 файлов разного размера (16, 32, 64 МБ) с приоритетами 1, 2, 4. Время приёма и передачи должно быть одинаковым для всех файлов.



Клиент №3

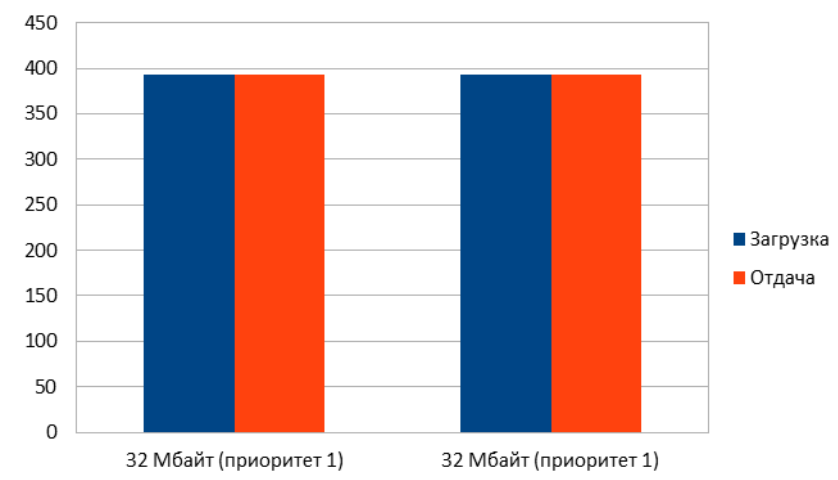


Клиент №6

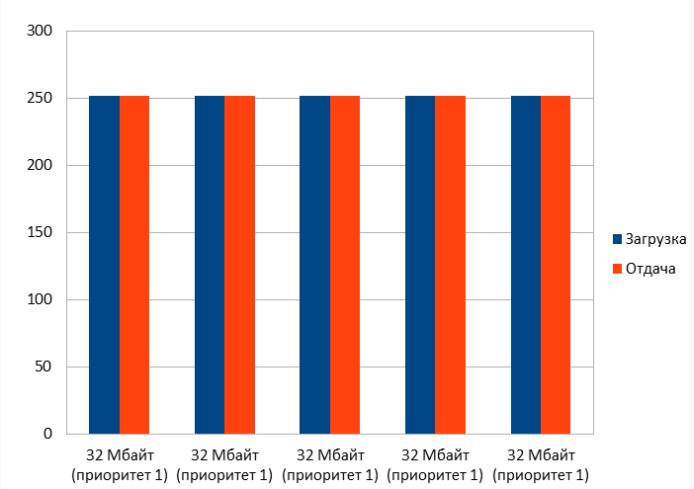
Вывод: файлы принялись и загрузились за одно и то же время, следовательно, приоритеты компенсировали размер файлов.

Тест №3

Аналог Теста №1, но приоритет у всех файлов – одинаковый.



Клиент №2

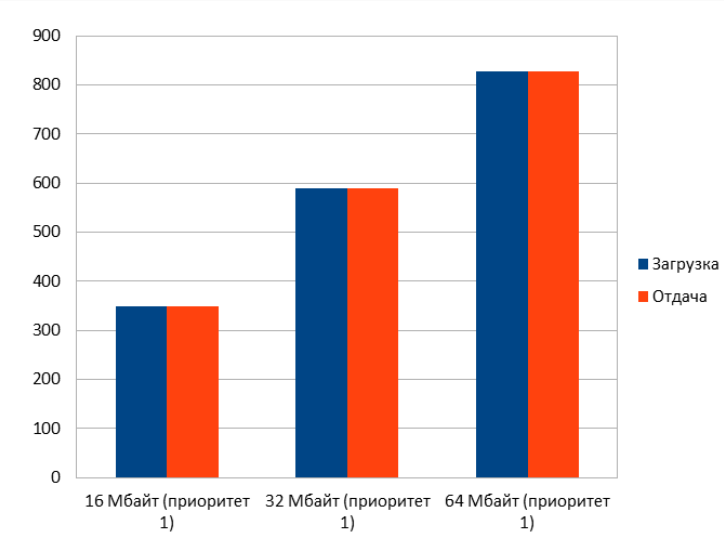


Клиент №5

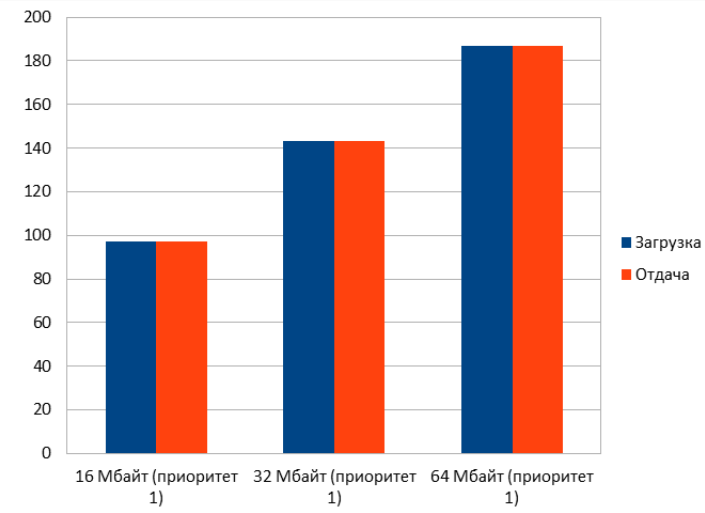
Вывод: файлы принялись и загрузились за одно и то же время, следовательно, приоритеты работают корректно.

Тест №4

Тест — аналогия Теста №2, только приоритет у всех загружаемых и выгружаемых файлов одинаковый. Данный тест предполагает, что файлы, которые участвуют в обмене должны загрузиться и выгрузиться за разное время.



Клиент №1

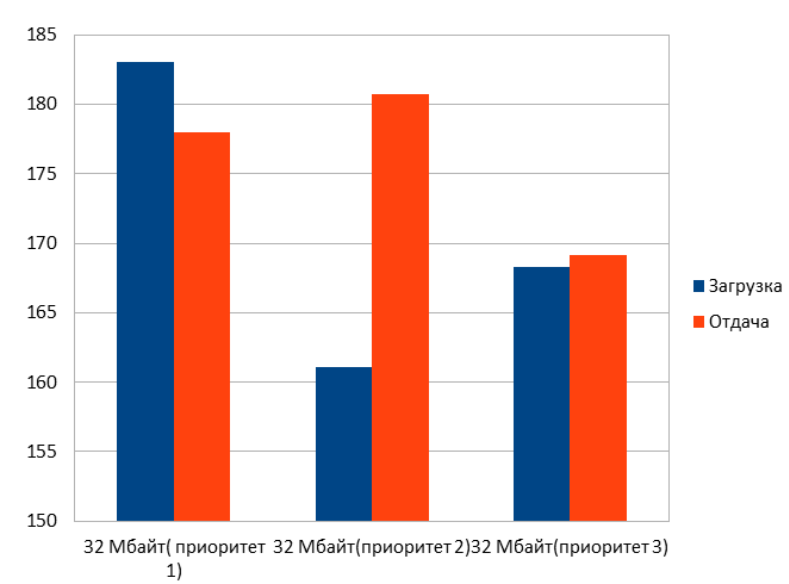


Клиент №3

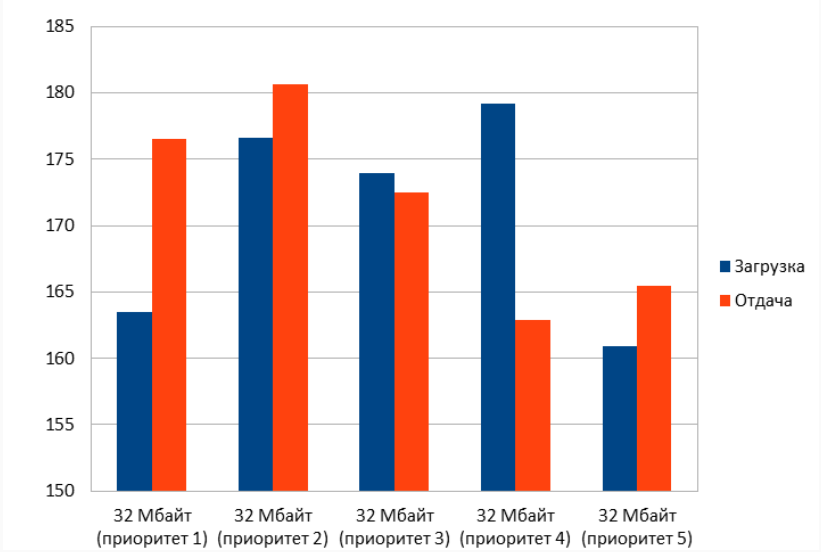
Вывод: файлы принялись и загрузились за разное время, следовательно, приоритеты сработали верно.

Тест №5

Данный тест — подобен Тесту №1, однако, в данном тесте обмен файлами будет производиться параллельно с методом разделения потока данных «по времени передачи». Предполагается, что файлы, имеющие больший приоритет, не всегда будут загружаться быстрее, чем те файлы, что имеют меньший приоритет.



Клиент №3



Клиент №7

Вывод: данный тест позволил увидеть «приоритетную несправедливость» при использовании метода разделения потока данных не по «блокам передачи», а «по времени передачи».

Резюме

Разработанное программное обеспечение

- Разработаны 2 алгоритма разделения потока данных при использовании FTP-протокола с учетом приоритетов. Выбран 1 («по кадрам передачи») для реализации;
- Написано 1700 строк кода для реализации функциональности клиентской части приложения (в этой части реализован алгоритм разделения потока данных) + около 700 строк кода для реализации графического интерфейса. ПО кроссплатформенное. Канал связи используется эффективно (без простоев);
- Разработан асинхронный FTP-сервер;
- Проведено около 10-ти тестов с использованием 14-ти реальных компьютеров и 1-го суперкомпьютера.

Оптимизация разделения потока данных

Разработанное программное обеспечение

Разработанный метод разделения потока данных соответствует всем предъявленным к нему требованиям, однако это не означает, что он является единственным.

Ещё одним возможным методом реализации является комбинация двух методов, а именно: метода разделения потока данных «по времени передачи» и «по блокам передачи». Например, необходимо передать N байт от файла, но, чтобы это не заняло более, чем S секунд. Такой метод разделения потока данных мог бы более «тонко» разделять поток данных, однако, требовал бы больших расходов для своей реализации.

Спасибо за внимание!