Лабораторная 1. Файловый менеджер

Цель работы

Основная цель решения задач из данного списка - освоить приемы работы с файлами и сетевым взаимодействием на низком уровне.

Порядок выполнения

Задания в этой лабораторной работе предусматривают индивидуальное самостоятельное выполнение. Преподаватель распределяет задания между студентами случайным образом.

На семинаре преподаватель может по запросу студентов пояснить задание в части требований к выполнению или стоящей за заданием математической концепцией.

Все задания подразумевают творческий подход к их выполнению. Задание описывает только общую канву программы. В процессе выполнения студент может столкнуться с рядом дизайнерских решений - выбор скорости анимации, размера элементов, цветового решения и так далее. Все это возлагается на студента как на разработчика программы.

Преподаватель вправе оценить удачные решения и снизить оценку за неудачные.

К задачам предлагаются дополнительные задания (пронумерованы), которые расширяют или углубляют данное задание. Студент может самостоятельно предложить расширение функционала программы. Дополнительные задания оцениваются преподавателем с учетом их сложности.

После самостоятельного выполнения задания преподаватель может попросить студента провести публичную демонстрацию выполненного задания, включая устные комментарии написанного хода и алгоритмических решений, примененных студентом при решении задачи.

Общие требования и рекомендации

В рамках данной лабораторной работы пользоваться нужно любыми самописными протоколами или же TCP/UDP, а пользоваться высокоуровневыми протоколами, как HTTP нельзя.

socket (нужно) - стандартная библиотека Python для работы с низкоуровневыми сетевыми соединениями. С её помощью можно создавать клиентские и серверные сокеты, устанавливать соединения, отправлять и принимать данные.

struct (нужно) - библиотека для работы со структурами данных. Она позволяет упаковывать и распаковывать данные в байты, что необходимо для создания и разбора сетевых пакетов.

select (опционально) - библиотека для мультиплексирования ввода/вывода. Она позволяет одновременно отслеживать несколько сетевых соединений, что полезно при создании серверов, способных обрабатывать множество клиентов одновременно.

asyncio (очень очень опционально) - библиотеки для многопоточности и асинхронного программирования соответственно. Они позволяют реализовывать параллельную обработку сетевых соединений и асинхронную передачу данных.

socketserver (в отчаянной ситуации) - библиотека, предоставляющая классы для создания сетевых серверов. Она абстрагирует многие низкоуровневые детали работы с сокетами и позволяет легко создавать TCP и UDP серверы.

scapy (опционально) - сторонняя библиотека для создания, отправки, перехвата и анализа сетевых пакетов. Она может быть полезна для более глубокого изучения сетевых протоколов и пакетов.

Варианты

Вариант 1.

Реализовать программу сервер, которая выполняет следующие функции:

- Использует модуль os для рекурсивного получения информации о всех файлах и директориях, начиная с папки, в которой была запущена программа.
- Сохраняет полученную информацию в файл формата JSON или XML.
- Работает аналогично консольным командам ls -alR в Linux или dir /s в Windows.
- Ожидает подключения клиентов и обрабатывает их запросы.
- При получении запроса от клиента на смену "корневой" директории, обновляет информацию о файлах для новой директории.
- Отправляет клиенту собранный файл с информацией о структуре директории по запросу.

Реализовать программу клиент, которая:

- Устанавливает сетевое соединение с сервером.
- Предоставляет пользовательский интерфейс для ввода команд.
- Позволяет пользователю отправлять серверу команду на установку новой "корневой" папки/директории.
- Запрашивает у сервера и получает файл с информацией о структуре выбранной директории.
- Отображает полученную информацию пользователю.

Дополнительные требования:

- Опционально: реализовать графический интерфейс для клиента, отображающий структуру директории в виде дерева.

Вариант 2.

Реализовать программу сервер, которая выполняет следующие функции:

- Хранит набор аудиофайлов в определенной директории.
- Использует модуль os для работы с файловой системой.
- При запуске создает и сохраняет в JSON-файл список всех аудиофайлов с их метаданными (имя файла, длительность, формат).
- Ожидает подключения клиентов и обрабатывает их запросы.
- Обрабатывает запросы на получение списка аудиофайлов.
- Вырезает указанный отрезок из аудиофайла, используя временные файлы.
- Отправляет клиенту запрошенный отрезок аудио.
- Обеспечивает многопоточную обработку для одновременной работы с несколькими клиентами.

Реализовать программу клиент, которая:

- Устанавливает сетевое взаимодействие с сервером.
- Позволяет пользователю выполнять следующие команды:
  - Получить список доступных аудиофалов.
  - Запросить отрезок аудиодорожки, указав имя файла, начальное и конечное время (в секундах)
- Получает от сервера запрошенный отрезок аудио и сохраняет его локально.

Дополнительные требования:

- Использовать библиотеку pydub для работы с аудиофайлами.
- Применять модуль tempfile для создания и управления временными файлами.
- Реализовать базовое логирование действий сервера и клиента.

Вариант 3.

Реализовать программу сервер, которая выполняет следующие функции:

- Использует модуль os для получения информации о всех процессах, запущенных на компьютере.
- Сохраняет полученную информацию в файл формата JSON или XML.
- Ожидает подключения от клиента и обрабатывает её запросы.
- При получении команды от клиента, обновляет информацию о процессах.
- Отправляет клиенту файл с обновленной информацией о процессах.
- Обрабатывает команды на отправку сигналов запущенным процессам.

Реализовать программу клиент, которая:

- Устанавливает сетевое соединение с сервером.
- Предоставляет пользовательский интерфейс для ввода команд.
- Позволяет пользователю отправлять серверу команду на обновление информации о процессах.
- Запрашивает у сервера и получает файл с обновленной информацией о процессах.
- Сохраняет полученные файлы в директории с форматом имени "./dd-mm-yyyy/hh:mm:ss.(json/xml)" для текущей даты и времени.
- Позволяет пользователю отправлять команды на отправку сигналов запущенным процессам (например, SIGTERM, SIGKILL).

Дополнительные требования:

- Реализовать механизм логирования действий обеих программ.

Вариант 4.

Реализовать программу сервер, которая выполняет следующие функции:

- Использует модуль os для получения информации о системных переменных окружения.
- Анализирует переменную окружения PATH для поиска директорий с исполняемыми файлами.
- Создает древовидную структуру данных, содержащую информацию о папках и находящихся в них исполняемых программах.
- Сохраняет полученную структуру в файл формата JSON или XML.
- Ожидает подключения от клиента и обрабатывает его запросы.
- При получении команды от клиента, обновляет информацию о программах и переменных окружения.

Отправляет клиенту файл с обновленной информацией.

- Позволяет устанавливать новые значения переменных окружения по команде от клиента.
- Реализовать программу клиент, которая:
- Устанавливает сетевое соединение с сервером.
- Предоставляет пользовательский интерфейс для ввода команд.
- Позволяет пользователю отправлять серверу команду на обновление информации о программах и переменных окружения.
- Запрашивает у сервера и получает файл с обновленной информацией.
- Отображает полученную информацию пользователю в удобном формате.
- Позволяет пользователю задавать новые значения переменных окружения для сервера.

Дополнительные требования:

- Реализовать возможность фильтрации и сортировки найденных программ по различным критериям (например, по имени, размеру, дате последнего изменения).
- Реализовать механизм логирования действий сервера и клиента.
- Обеспечить безопасную передачу данных между сервером и клиентом, особенно при установке новых значений переменных окружения.

- Реализовать возможность сохранения истории изменений переменных окружения на сервере.

Вариант 5.

Реализовать программу сервер, которая выполняет следующие функции:

- Принимает названия программ в качестве аргументов при запуске.
- Создает папку/директорию для каждой переданной программы с соответствующим именем.
- Циклически запускает каждую программу с интервалом в 10 секунд, используя модуль os.
- Записывает стандартный вывод каждой запущенной программы в новый файл в соответствующей папке.
- При запуске загружает информацию о ранее запущенных программах из файла формата JSON/XML.
- Добавляет новые программы к списку уже известных программ.
- Перед завершением работы сохраняет информацию о всех программах, папках и файлах в файл формата JSON/XML.
- Ожидает подключения от клиента и обрабатывает его запросы.
- Позволяет добавлять новые программы в список запускаемых программ по команде от клиента.
- Формирует и отправляет клиенту файл, содержащий объединенный вывод всех запусков указанной программы.

Реализовать программу клиент, которая:

- Устанавливает сетевое соединение с сервером.
- Предоставляет пользовательский интерфейс для ввода команд.
- Позволяет пользователю отправлять серверу команду на добавление новой программы в список запускаемых программ.
- Запрашивает у сервера и получает файл с объединенным выводом для указанной программы.
- Отображает полученную информацию пользователю в удобном формате.

Дополнительные требования:

- Реализовать механизм безопасного добавления новых программ в список запускаемых (проверка на существование, права доступа и т.д.).
- Реализовать механизм логирования действий сервера и клиента.
- Реализовать возможность настройки интервала запуска программ (вместо фиксированных 10 секунд).
- Обеспечить возможность graceful shutdown сервера с корректным завершением всех запущенных процессов.

Вариант 6.

Реализовать программу сервер, которая выполняет следующие функции:

- Хранит CSV файлы в папках, где имя папки соответствует названию таблицы.
- Обрабатывает SELECT запросы от клиента с возможностью одного условия WHERE.
- Анализирует полученный запрос и выполняет соответствующую выборку данных из CSV файлов.
- Формирует и отправляет клиенту результат запроса в формате CSV.
- По запросу клиента формирует и отправляет JSON-файл со структурой всех таблиц и их колонок.
- Поддерживает базовые операции сравнения в условии WHERE (=, <, >, <=, >=, !=).
- Обеспечивает параллельную обработку запросов от нескольких клиентов.

Реализовать программу клиент, которая:

- Устанавливает сетевое соединение с сервером.

- Предоставляет пользовательский интерфейс для ввода SQL-подобных запросов.
- Отправляет серверу SELECT запросы с возможностью одного условия WHERE.
- Получает от сервера и отображает результат запроса в формате CSV.
- Позволяет пользователю запросить у сервера JSON-файл со структурой всех таблиц и их колонок.
- Отображает полученную структуру таблиц в удобном для пользователя формате.

Дополнительные требования:

- Применять модуль tempfile для создания и управления временными файлами.
- Реализовать парсер SQL-подобных запросов на стороне сервера для обработки SELECT и WHERE условий.
- Использовать модуль csv для работы с CSV файлами на сервере.
- Реализовать механизм кэширования на сервере для оптимизации частых запросов.
- Обеспечить корректную обработку ошибок и исключений, особенно при работе с файловой системой и парсинге запросов.
- Реализовать механизм логирования действий сервера и клиента.
- Использовать объектно-ориентированный подход при разработке программ.
- Обеспечить безопасную передачу данных между сервером и клиентом.
- Реализовать базовую аутентификацию клиентов на сервере.
- Предусмотреть возможность расширения функциональности, например, добавление поддержки более сложных SQL запросов в будущем.