Лабораторная работа №5: "Обмен данными по именованным каналам"

Тема: Организация клиент-серверного взаимодействия между процессами с помощью именованных каналов (Named Pipes).

Цель работы: Изучить механизм именованных каналов для организации двустороннего обмена структурированными данными между независимыми процессами. Научиться проектировать протокол взаимодействия и обрабатывать жизненный цикл клиент-серверной системы.

Современные веб-страницы часто выполняют сложные вычисления (например, фильтрация изображений, анализ данных, 3D-графика). Если выполнять их в основном потоке браузера, интерфейс "зависнет". Для решения этой проблемы существуют **Web Workers** — фоновые процессы, которым основной поток может отправлять задачи и получать от них результаты, не блокируя пользовательский интерфейс.

В этой работе смоделируем именно такую систему:

- `Browser.exe` — управляющий процесс, который играет роль диспетчера задач для браузера. Он отвечает за распределение работы между фоновыми процессами.
- `Worker.exe` — фоновый вычислительный процесс (Web Worker), который получает задачи, "тяжело" работает и отправляет результат обратно.

Взаимодействие между ними будет осуществляться через **именованные каналы**, которые идеально подходят для такой коммуникации.

Роль программы `Browser.exe` (Основной поток браузера, "Сервер")

1. При запуске `Browser.exe` запрашивает у пользователя:
   - `N` — количество `Worker`-процессов, которые нужно запустить в "пуле" (например, 2).
   - `M` — общее количество задач, которые нужно обработать (например, 10).
2. Для каждого `Worker`-а (от 0 до N-1) `Browser.exe` создает **два именованных канала**:
   - Один для отправки задач *в* `Worker`: `\\.\pipe\worker_in_{ID}`
   - Второй для получения результатов *от* `Worker`-а: `\\.\pipe\worker_out_{ID}`

   (где {ID} — это номер воркера, например, 0, 1, ...)
3. Запускает `N` экземпляров процесса `Worker.exe`. Каждому процессу через аргументы командной строки передается его уникальный ID.
4. Распределяет все `M` задач между доступными `Worker`-ами. Например, поочередно отправляет задачу свободному воркеру, дожидается ответа, а затем отправляет ему следующую задачу из очереди.

5. **Протокол обмена:**
    - Для отправки задачи `Browser` формирует структуру, содержащую тип задачи и данные, и записывает ее в соответствующий канал `worker_in`.
    - После отправки задачи `Browser` блокируется в ожидании ответа, читая данные из канала `worker_out`.
6. После того как все `M` задач выполнены и результаты получены, `Browser` должен корректно завершить работу `Worker`-ов. Для этого он отправляет каждому `Worker`-у специальную "команду-завершения" (например, задачу с особым типом).
7. Дожидается завершения всех дочерних процессов `Worker.exe`.
8. Корректно закрывает все дескрипторы каналов и завершает работу.

Роль программы `Worker.exe` (Фоновый процесс, "Клиент")

1. При запуске `Worker.exe` получает свой ID из аргументов командной строки.
2. На основе ID формирует имена своих каналов (`worker_in_{ID}` и `worker_out_{ID}`) и подключается к ним.
3. Входит в основной рабочий цикл:
   a. Ожидает и читает из своего входного канала (`worker_in`) структуру с очередной задачей.
   b. Проверяет тип задачи. Если это "команда-завершения", выходит из цикла и переходит к завершению работы.
   c. Если это обычная задача, **выполняет вычисления согласно своему индивидуальному варианту**.
   d. Формирует структуру с результатом и отправляет ее обратно в `Browser` через свой выходной канал (`worker_out`).
4. После выхода из цикла корректно закрывает все дескрипторы и завершает свою работу.

Индивидуальные варианты

Задача, которую `Worker.exe` выполняет над полученным массивом данных(если лень или неинтересно - можно поспать 😴😴😴):

1. **Фильтр "Сепия"**: Для каждого числа в массиве (имитация пикселя) применяет формулу `R*0.393 + G*0.769 + B*0.189`.
2. **Поиск простых чисел**: Находит все простые числа в заданном диапазоне (данные - `{min, max}`).
3. **Сортировка слиянием**: Сортирует полученный массив чисел.
4. **Вычисление CRC32**: Считает контрольную сумму для массива байт.
5. **Обработка статистики**: Находит медиану и стандартное отклонение для массива чисел.

6. **Шифрование XOR**: Шифрует массив данных с помощью XOR-ключа (ключ передается вместе с данными).
7. **Поиск подстроки**: Находит количество вхождений одной строки в другую (передаются как массивы `char`).
8. **Матричное умножение**: Умножает две небольшие матрицы, переданные в виде одного массива.
9. **Вычисление факториала**: Находит факториал для каждого числа в массиве.
10. **Построение гистограммы**: Считает, сколько раз каждое число встречается в массиве.
11. **Преобразование Фурье**: Выполняет упрощенное дискретное преобразование Фурье над массивом.
12. **Сжатие данных (RLE)**: Сжимает массив данных по алгоритму Run-Length Encoding.
13. **Поиск пути в графе**: Находит кратчайший путь в простом графе, заданном матрицей смежности.
14. **Инверсия изображения**: Для каждого числа в массиве (пикселя) вычисляет `255 - pixel`.

Задание по тестированию программы

Группа 1: Позитивные тесты (проверка базовой логики)

- **Тест 1.1: "Один воркер, одна задача"**
  - **Сценарий:** Запустить систему с 1 `Browser` и 1 `Worker`. Отправить одну простую задачу (например, отсортировать массив `[5, 2, 8]`).
  - **Проверка:** Убедиться, что полученный от `Worker`-а результат корректен (массив `[2, 5, 8]`).
- **Тест 1.2: "Пул воркеров, много задач"**
  - **Сценарий:** Запустить систему с 2 `Worker`-ами и 10 задачами.
  - **Проверка:** Убедиться, что все 10 задач были выполнены и все 10 результатов верны. Проверить, что `Browser` корректно дождался завершения всех `Worker`-ов.
- **Тест 1.3: "Корректное завершение"**
  - **Сценарий:** Запустить систему, выполнить несколько задач, а затем отправить команду на завершение.
  - **Проверка:** Убедиться, что все процессы `Worker.exe` корректно завершили свою работу (не "зависли").

Группа 2: Негативные и граничные тесты

- **Тест 2.1: "Пустая задача"**
  - **Сценарий:** `Browser` отправляет `Worker`-у задачу с пустым массивом данных.
  - **Ожидаемое поведение:** `Worker` не должен падать. Он должен корректно обработать этот случай и вернуть соответствующий результат (например, пустой

массив).

- **Проверка:** Тест **проходит**, если `Worker` вернул ожидаемый "пустой" результат и не завершился аварийно.