Opis Systemu Rozproszonego

System składa się z trzech oddzielnych programów:

---

1. Server (`server.py`)

- **Funkcjonalność:**
  - Uruchamia managera, który tworzy i udostępnia dwie kolejki:
    - `qData` – do przesyłania danych (zadań) z klienta do workerów.
    - `qResults` – do przesyłania wyników (częściowych obliczeń) z workerów do klienta.
- **Działanie:**
  - Serwer działa stale (metoda `serve_forever()`) i czeka na połączenia od klienta oraz workerów.

---

2. Client (`client.py`)

- **Funkcjonalność:**
  - Wczytuje dane macierzy i wektora z plików.
  - Dzieli macierz na fragmenty (zadania) – liczba fragmentów (granularność) ustalana jest przez argument programu (`<liczba_zadań>`).
  - Wysyła każde zadanie (fragment macierzy wraz z wektorem i indeksem startowym) do kolejki `qData`.
  - Po wysłaniu wszystkich zadań, wysyła do kolejki tyle "poison pills" (wartość `None`), ile jest zadań, aby każdy worker otrzymał sygnał zakończenia.
  - Odbiera częściowe wyniki z kolejki `qResults` i scala je w końcowy wynik, który następnie wyświetla.
- **Zakończenie:**
  - Client kończy pracę, gdy otrzyma wszystkie częściowe wyniki.

---

3. Worker (`worker.py`)

- **Funkcjonalność:**
  - Łączy się z managerem, pobierając kolejki `qData` oraz `qResults`.
  - Na podstawie argumentu (`<liczba_zadań>`) uruchamia określoną liczbę podprocesów.
- **Działanie podprocesów:**
  - Każdy podproces w pętli:
    - Pobiera zadanie z kolejki `qData` (metoda `get()` – blokująca).
    - Wykonuje operację mnożenia macierzy przez wektor (funkcja `multiply`).
    - Po otrzymaniu wartości `None` (poison pill), przerywa pętlę i kończy pracę.
- **Zakończenie:**
  - Worker kończy działanie, gdy każdy z podprocesów odbierze poison pill.

Kolejność Uruchamiania

1. **Serwer:**
   - Musi być uruchomiony jako pierwszy, ponieważ tworzy i udostępnia kolejki, z którymi będą się komunikować klient i workerzy.
2. **Worker:**
   - Może być uruchomiony po serwerze. Jeśli uruchomiony przed klientem, będzie blokował się, czekając na zadania w kolejce.
   - Kolejność między workerami a klientem nie jest krytyczna – o ile serwer jest aktywny, workerzy będą czekać na dane.
3. **Client:**
   - Najlepiej uruchomić po serwerze, a także po workerach (choć niekoniecznie – workery będą czekać, jeśli zadania nie pojawią się natychmiast).

Zależności Między Liczbami Procesów i Zadaniami

- **Liczba zadań:**
  - Ustalana jest przez klienta jako argument `<liczba_zadań>`.
  - Określa, na ile fragmentów zostanie podzielona macierz.
  - Klient wysyła dokładnie tyle poison pills (`None`), ile jest zadań, aby każdy worker (podproces) otrzymał sygnał zakończenia.
- **Liczba workerów:**
  - W `worker.py` liczba podprocesów jest ustawiana na wartość `<liczba_zadań>`.
  - Jeśli liczba wysłanych poison pills będzie mniejsza niż liczba uruchomionych workerów, niektóre workery mogą pozostać w nieskończonej pętli oczekując na zadania.
  - Jeśli liczba poison pills jest równa liczbie workerów, każdy worker odbierze jeden pill i poprawnie zakończy pracę.

Przebieg Działania oraz Warunki Zakończenia

Serwer

- **Działanie:**
  - Uruchamia managera i czeka w nieskończoność na połączenia od klienta i workerów.
- **Zakończenie:**
  - Działa stale (metoda `serve_forever()`).

Client

- **Działanie:**
  - Wczytuje dane z plików.

- Dzieli macierz na fragmenty (według podanej liczby zadań).
- Wysyła zadania do kolejki `qData`.
- Po wysłaniu zadań, wysyła tyle poison pills, ile jest zadań.
- Odbiera częściowe wyniki z `qResults` i scala je w końcowy wynik.
- Wyświetla wynik i czas wykonania.

- **Zakończenie:**
  - Client kończy pracę, gdy otrzyma wszystkie częściowe wyniki.

Worker

- **Działanie:**
  - Łączy się z managerem i pobiera kolejki.
  - Uruchamia określoną liczbę podprocesów.
  - Każdy podproces w pętli:
    - Pobiera zadanie (blokująco – oczekuje na dane).
    - Wykonuje mnożenie i wysyła wynik do `qResults`.
    - Po otrzymaniu poison pill (`None`), przerywa pętlę i kończy pracę.
- **Zakończenie:**
  - Worker kończy działanie, gdy każdy z podprocesów odbierze poison pill.

Co Można, a Czego Nie Można

Co można:

- Uruchomić serwer na jednej maszynie, a klienta i workerów na innych maszynach, o ile mogą się one połączyć (np. przez sieć) i używają tych samych adresu oraz klucza autoryzacyjnego.
- Klient i workerzy mogą być uruchamiani w dowolnej kolejności, pod warunkiem, że serwer jest aktywny.
- Dostosować liczbę zadań (granularność) przez podanie odpowiedniej liczby jako argument klienta, co wpływa na podział pracy i liczbę uruchamianych workerów (podprocesów).

Czego nie można (lub należy unikać):

- Uruchamiać klienta lub workerów przed uruchomieniem serwera – wtedy nie uda się nawiązać połączenia.
- Wysyłać nieodpowiedniej liczby poison pills –
  - Jeśli będzie ich za mało, część workerów będzie blokować się w nieskończonej pętli.
  - Jeśli będzie ich za dużo, może dojść do nadmiarowych odebrań (choć przy poprawnej synchronizacji kolejki nie powinno to wpływać na wynik).
- Ustalać liczbę workerów niezależnie od liczby wysłanych poison pills – w tym systemie liczba podprocesów workerów musi być zgodna z liczbą poison pills wysyłanych przez klienta, aby zapewnić poprawne zakończenie wszystkich podprocesów.

Podsumowanie

- **Serwer:**
  - Uruchamiany jako pierwszy, stale dostępny, udostępnia kolejki.
- **Client:**
  - Dzieli dane na zadania według podanej liczby.
  - Wysyła zadania oraz poison pills.
  - Scala wyniki i wyświetla końcowy rezultat.
- **Worker:**
  - Uruchamia podprocesy zgodnie z liczbą zadań.
  - Wykonuje obliczenia.
  - Kończy działanie po odebraniu poison pill przez każdy podproces.
- **Kolejność uruchamiania:**
  - **Serwer → (Worker i/lub Client)**
  - Zaleca się, aby serwer był uruchomiony jako pierwszy, a następnie workerzy i klient.
- **Zależności:**
  - Liczba zadań klienta determinuje zarówno granularność danych, jak i liczbę workerów oraz poison pills.
  - Poprawne dopasowanie tych wartości jest kluczowe, aby uniknąć infinite loopów oraz niezamkniętych procesów.