Take home assignment

Завдання: Блокуючі запити, статус завершення, мова реалізації: C++. У якості носіїв компонентів обчислення розглядаються процеси.

1. Опис менеджера обчислень (Manager)

Функціональність:
Менеджер обчислень (клас `Manager`) керує групами обчислювальних компонентів (процесів), які виконуються незалежно один від одного. Він відповідає за створення групи, додавання компонентів, їх запуск, моніторинг статусу та збір результатів.

Ключові функції:

- **createGroup**: Створення нової групи обчислень.
- **addComponent**: Додавання нового компонента (обчислення) до поточної групи.
- **run**: Запуск усіх доданих компонентів.
- **status**: Перевірка статусу всіх активних компонентів.
- **summary**: Перегляд результатів обчислень всіх компонентів.
- **clear**: Очищення всіх активних компонентів та групи.

```
// Менеджер групи
class Manager {
public:
    Manager() {
        signal(SIGINT, signalHandler);
    }
```

2. Створення групи

Функціональність:

- **`createGroup`** дозволяє створити нову групу обчислень. Група є основною одиницею, в рамках якої зберігаються всі компоненти (функції обчислень), що будуть виконуватися паралельно.

```
// Створити групу
void createGroup(const std::string& name) {
    if (currentGroup.empty()) {
        currentGroup = name;
        std::cout << "Group " << name << " created.\n";
    } else {
        std::cerr << "A group already exists: " << currentGroup << "\n";
    }
}
```

3. Додавання нового компонента

Функціональність:

- `addComponent` додає новий компонент до поточної групи обчислень, створюючи окремий процес для виконання функції, яка передається як аргумент. Кожен компонент виконується у своєму окремому процесі, і результат обчислення передається через канал (pipe) для подальшого зчитування батьківським процесом.

```
// Додати новий компонент
void addComponent(const std::string& name, FunctionType func, int arg) {
    if (currentGroup.empty()) {
        std::cerr << "No group created yet. Please create a group first.\n";
        return;
    }

    int pipefd[2];
    if (pipe(pipefd) == -1) {
        perror("pipe");
        return;
    }

    pid_t pid = fork();
    if (pid == -1) {
        perror("fork");
        return;
    }

    if (pid == 0) { // Дочірній процес
        close(pipefd[0]); // Закриваємо читання
        double result = func(arg); // Виконуємо обчислення
        write(pipefd[1], &result, sizeof(result)); // Надсилаємо результат
        close(pipefd[1]);
        exit(0); // Завершення
    } else { // Батьківський процес
        close(pipefd[1]); // Закриваємо запис
        components[name] = pid; // Зберігаємо PID
        pipes[name] = pipefd[0]; // Зберігаємо pipe
        activeProcesses[pid] = name; // Додаємо до активних процесів
    }
}
```

4. Запуск обчислень (run)

Функціональність:

- `run` виконує запуск усіх компонентів, що були додані до поточної групи. Якщо група ще не була створена, виводиться повідомлення про помилку. Якщо група існує, метод ініціює виконання кожного компоненту і виводить інформацію про поточний статус кожного з них.

```
void run() {
    if (currentGroup.empty()) {
        std::cerr << "No group created yet. Please create a group first.\n";
        return;
    }

    std::cout << "Running group " << currentGroup << "...\n";

    for (const auto& [name, pid] : components) {
        std::cout << "Component " << name << " is running (PID: " << pid << ").\n";
    }
}
```

5. Підсумок (summary)

Функціональність:

- `summary()` відповідає за виведення результатів обчислень усіх компонентів, які належать поточній групі. Якщо група ще не була створена, метод виводить повідомлення про помилку. Інакше, він переглядає зібрані результати для кожного компонента та виводить їх у консоль.

```
void summary() const {
    if (currentGroup.empty()) {
        std::cerr << "No group created yet. Please create a group first.\n";
        return;
    }

    for (const auto& [name, result] : results) {
        std::cout << "Component " << name << " result: " << result << "\n";
    }
}
```

6. Очищення

Функціональність:

- `clear` відповідає за очищення всіх активних компонентів і звільнення ресурсів, пов'язаних з поточними обчисленнями. Він видаляє всі компоненти, канали для передачі даних, результати виконання, а також зупиняє всі активні процеси. Також цей метод очищає дані про поточну групу, що використовувалася для обчислень. Після виконання методу на екран виводиться повідомлення, яке підтверджує, що всі компоненти та група були очищені.

```
void clear() {
    components.clear();
    pipes.clear();
    results.clear();
    activeProcesses.clear();
    currentGroup.clear();  // Очищаємо групу
    std::cout << "Cleared all components and group.\n";
}
```

7. Функції, які виконують математичні операції для різних значень та застосовуються для обчислень у рамках окремих компонентів:

```
// Функція для обчислення x * x * 1.5
double function_g(int x) {
    return x * x * 1.5;
}

// Функція для обчислення квадратного кореня з x
double function_h(int x) {
    return std::sqrt(x);
}

// Функція для обчислення куба x
double function_f(int x) {
    return x * x * x;
}
```

Команди:

```cpp
if (tokens[0] == "help") {
    std::cout << "Available commands:\n";
    std::cout << "  group <name>        - Create a new task group (only one group is allowed).\n";
    std::cout << "  new <component_name> <arg> - Add new component to the current group with argument.\n";
    std::cout << "  run                 - Run all added components.\n";
    std::cout << "  status              - Show status of active components.\n";
    std::cout << "  summary             - Show results of all components.\n";
    std::cout << "  clear               - Clear all active components and group.\n";
    std::cout << "  exit                - Exit the program.\n";
```

Приклад використання:

```
/Users/viktoriyabilyk/CLionProjects/takehome_os/cmake-build-debug/ComputationManager
Command-line interface started. Type 'help' for commands.
> help
Available commands:
  group <name>        - Create a new task group (only one group is allowed).
  new <component_name> <arg> - Add new component to the current group with argument.
  run                 - Run all added components.
  status              - Show status of active components.
  summary             - Show results of all components.
  clear               - Clear all active components and group.
  exit                - Exit the program.
> group g1
Group g1 created.
> new g 3
> new h 4
> new f 5
> run
Running group g1...
Component f is running (PID: 67304).
Component g is running (PID: 67297).
Component h is running (PID: 67299).
> status
Component g finished successfully, result: 13.5
Component h finished successfully, result: 2
Component f finished successfully, result: 125
No active components are running.
> summary
Component f result: 125
Component g result: 13.5
Component h result: 2
> clear
Cleared all components and group.
> exit
Exiting.
Program terminated. Goodbye!
```