ДЗ 3: exec-graph

Дата выдачи: 29.10.24

Дедлайн сдачи: 26.11.24

Описание

Требуется реализовать на языке Си программу, позволяющую запускать на исполнение задачи, описанные в конфигурационном файле.

Максимальный балл за задание: **20**.

Функциональные требования

- Задачи описываются в конфигурационном файле (см. примеры)
  - Секция **`[main]`** описывает общие настройки для запускаемых задач
    - **`max_concurrent_tasks`**: максимальное число параллельно запущенных задач; по умолчанию равно 3
    - **`default_timeout`**: таймаут по умолчанию (в секундах) на выполнение каждой из задач; по умолчанию равно 10
    - (!) эта секция опциональна, но присутствует в конфиге не более одного раза
  - Секция **`[task]`** описывает
    - **`name`**: название задачи; обязательное поле, уникально среди всех задач
    - **`requires`**: список имён задач, разделённых пробелом, после окончания которых текущая задача может начать выполнение; если отсутствует или содержит единственную строку **`none`**, то зависимостей нет; возможны дубликаты (кроме случая с **`none`**)
    - **`timeout`**: таймаут (в секундах) на выполнение задачи, 0 означает отсутствие таймаута; если отсутствует, то используется значение из секции **`[main]`**
    - **`type`**: тип задачи (**`SLEEP`** или **`EXEC`**)
    - **`sleep_duration`**: длительность сна (в секундах); для задач типа EXEC игнорируется
    - **`exec_command`**: команда для запуска; для задач типа **`SLEEP`** игнорируется
    - (!) эта секция может не встречаться ни разу
  - Секции идут в произвольном порядке и отделяются друг от друга одной или несколькими пустыми строками, в конце и начале конфига может быть произвольное число пустых строк (в том числе ноль)
  - Комментарии – строки, начинающиеся на символ **#** – игнорируются
  - Необходимы всевозможные проверки типов и значений всех аргументов и секций (их названий и значений), а также проверка ацикличности графа зависимостей задач

- Программа должна обрабатывать следующие аргументы командной строки
  - **`--config (-c)`**: путь к конфигурационному файлу с описанием исполняемых задач
  - **`--log (-l)`**: путь к директории для логов исполняемых задач
  - **`--verbose (-v)`**: включить отрисовку (поддерживаются значения **`none`**, **`table`** и **`graph`**); опция **`-v`** без значения означает **`-v table`**, отсутствие опции **`-v`** означает **`-v none`**
  - **`--render-latency (-r)`**: интервал между двумя последовательными отрисовками

Аргументы могут передаваться в произвольном порядке. Если переданы неизвестные или некорректные аргументы командной строки, программа должна сообщать пользователю причину ошибки (в stderr) и завершаться.

- Программа должна состоять из нескольких модулей (допустимо добавление любых необходимых заголовочных файлов, а также файлов с реализацией, помимо приведённых в шаблоне)

- Сборка программы должна осуществляться с помощью Makefile
  - Опции компилятора обязательно должны включать в себя следующие: **`-Wall, -Werror, -std=c11, -O0 / -O2`**
  - Должны быть поддержаны 4 режима сборки: **`release`** (по умолчанию), **`debug`**, **`test`**, **`valgrind`**
  - В режиме **`debug`** сборки выключены оптимизации (**`-O0`**), в режимах **`release`**, **`test`** и **`valgrind`** – включены (**`-O2`**)
  - Режим **`valgrind`** собирает тесты и запускает их командой

```
valgrind --leak-check=full <test-executable>
```

  - Режим **`test`** собирает тесты (вместе с бенчмарками), запускает их, определяет степень покрытия исходного кода тестами, запускает линтер **`clang-tidy`**
  - Тесты должны покрывать все основные краевые случаи использования тестируемых функций

Стилистические требования

- Комбинация styleguide + clang-tidy + **`-Wall -Werror`** остаётся в силе

- Константы можно оформлять в enum'ы (тогда допускается `CAPS_SNAKE_CASE`) и/или в отдельные статические переменные (тогда используйте только венгерскую нотацию: `kMyLovelyConst`)

Дополнительно

Task status rendering (3 + 7 баллов)

Реализуйте динамически обновляемое отображение статусов всех задач в виде таблички. Рендеринг должен производиться в отдельном потоке мастер-процесса. Остальные подробности ищите в шаблоне.

Для самых храбрых – отрисовка статусов в виде графа зависимостей из конфига.

Pipeline graph (10 + 10 баллов)

Сейчас задачи никак между собой не связаны: выдача одних задач не подаётся на вход другим. Теоретически возможно, что более поздняя задача, зависящая от другой, будет обрабатывать её выход, записанный в файл с логом. Но для этого исходная задача должна полностью завершиться, успешно записав файл с логом.

Помимо графа зависимостей задач, имеет смысл также описать граф передачи данных между задачами, запущенными параллельно. Тогда данные будут поступать из первого процесса во второй почти сразу, в процессе их генерации, а не после полного дампа в файл с логом. Не забудьте провалидировать граф передачи данных: каждое его ребро должно связывать два процесса, которые могут быть запущены параллельно.

В базовом варианте этой задачи предполагается, что

- у каждого процесса выход направляется на вход не более чем одному процессу;
- каждому процессу на вход поступают данные не более чем из одного процесса.

Более сложный вариант этой задачи снимает одно или оба из этих ограничений. В случае снятия второго ограничения имеет смысл требовать строгое упорядочение данных (например, в процессе возрастания номеров задач), чтобы не получалась каша на входе.