Визуализация и поиск дефектов в многозадачных программах на языке Python в интегрированной среде разработки PyCharm

Шашкова Елизавета, 544 группа
Научный руководитель: к. ф.-м. н. Булычев Д.Ю.
Рецензент: Власовских А.С., JetBrains

Пример многозадачной программы

```
def produce():
    for num in count():
        sleep(1)
        queue.put(num)

def consume():
    while True:
        num = queue.get()

Thread(target=produce, name="Producer").start()
Thread(target=consume, name="Consumer-1").start()
Thread(target=consume, name="Consumer-2").start()
```

Схема работы программы

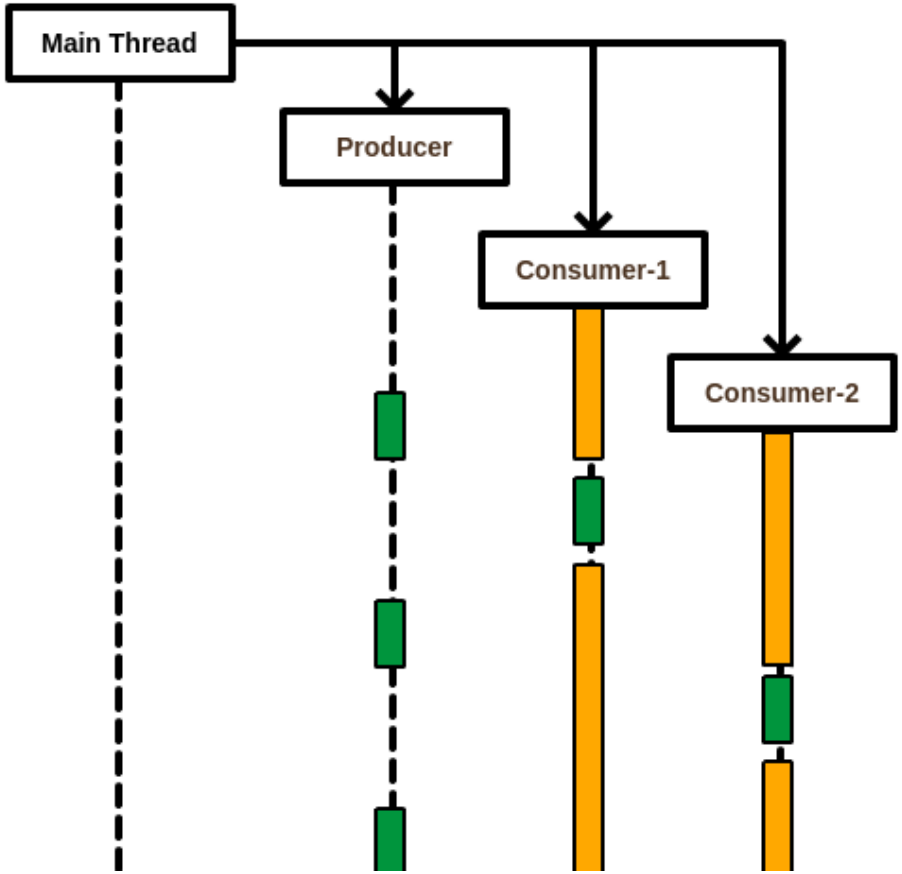


Рис. 1. Пример схемы, описывающей ход выполнения многозадачной программы.

Цель

- Упростить понимание хода выполнения многозадачных программ на языке Python

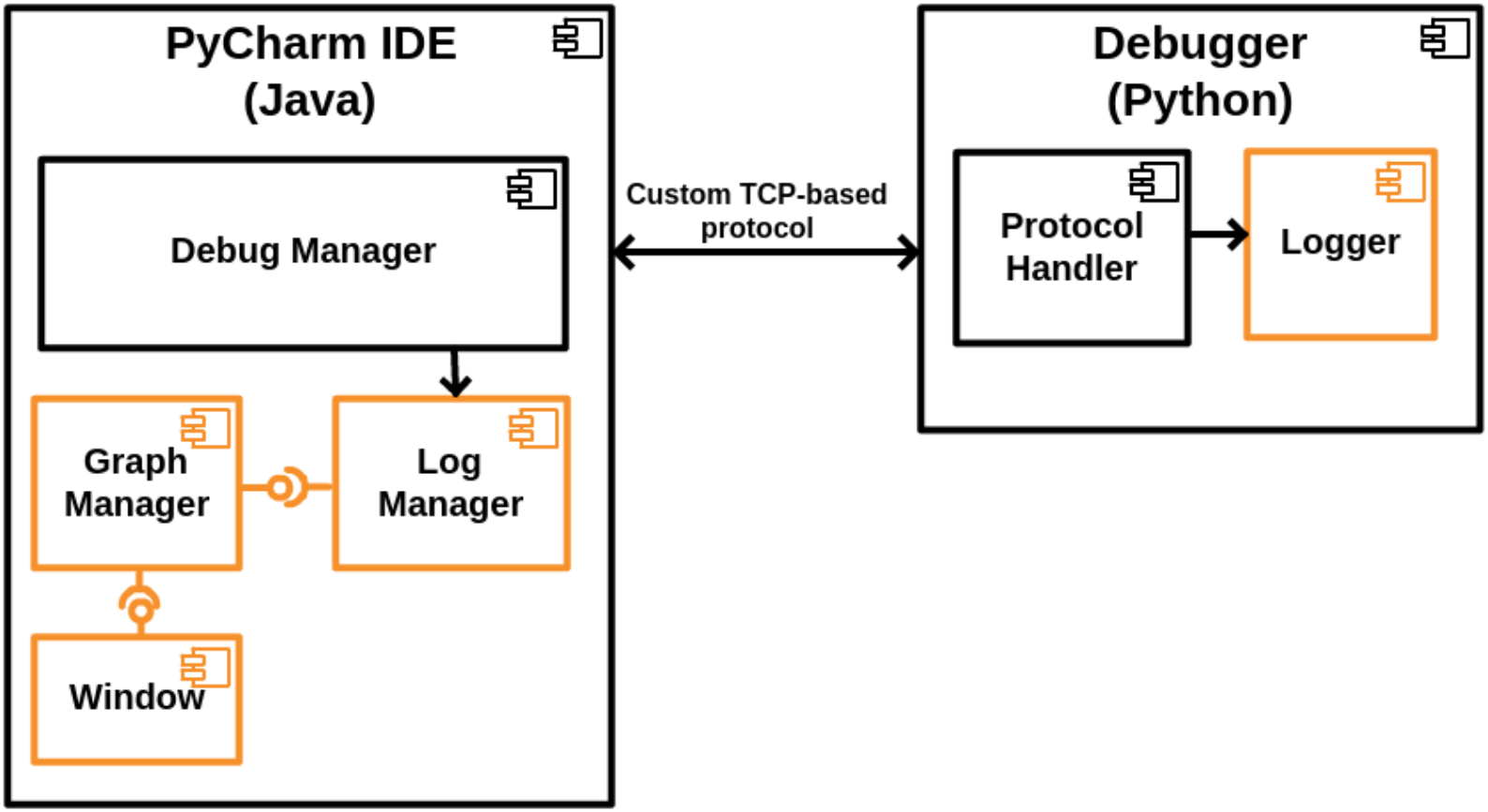
Многозадачность в Python

- threading
  - Интерфейс к потокам ОС
- asyncio
  - Легковесные задачи асинхронного ввода-вывода

Задачи

- Реализовать протоколирование событий в многозадачной программе на языке Python, запущенной в режиме отладки в IDE PyCharm
- Реализовать расширение к PyCharm, осуществляющее
  - визуализацию исполнения многозадачной программы
  - навигацию к исходному коду
- Реализовать поиск дефектов:
  - неиспользованные объекты синхронизации
  - потоки, не дождавшиеся объединения
  - взаимные блокировки (deadlock)
- Провести тестирование полученного инструмента

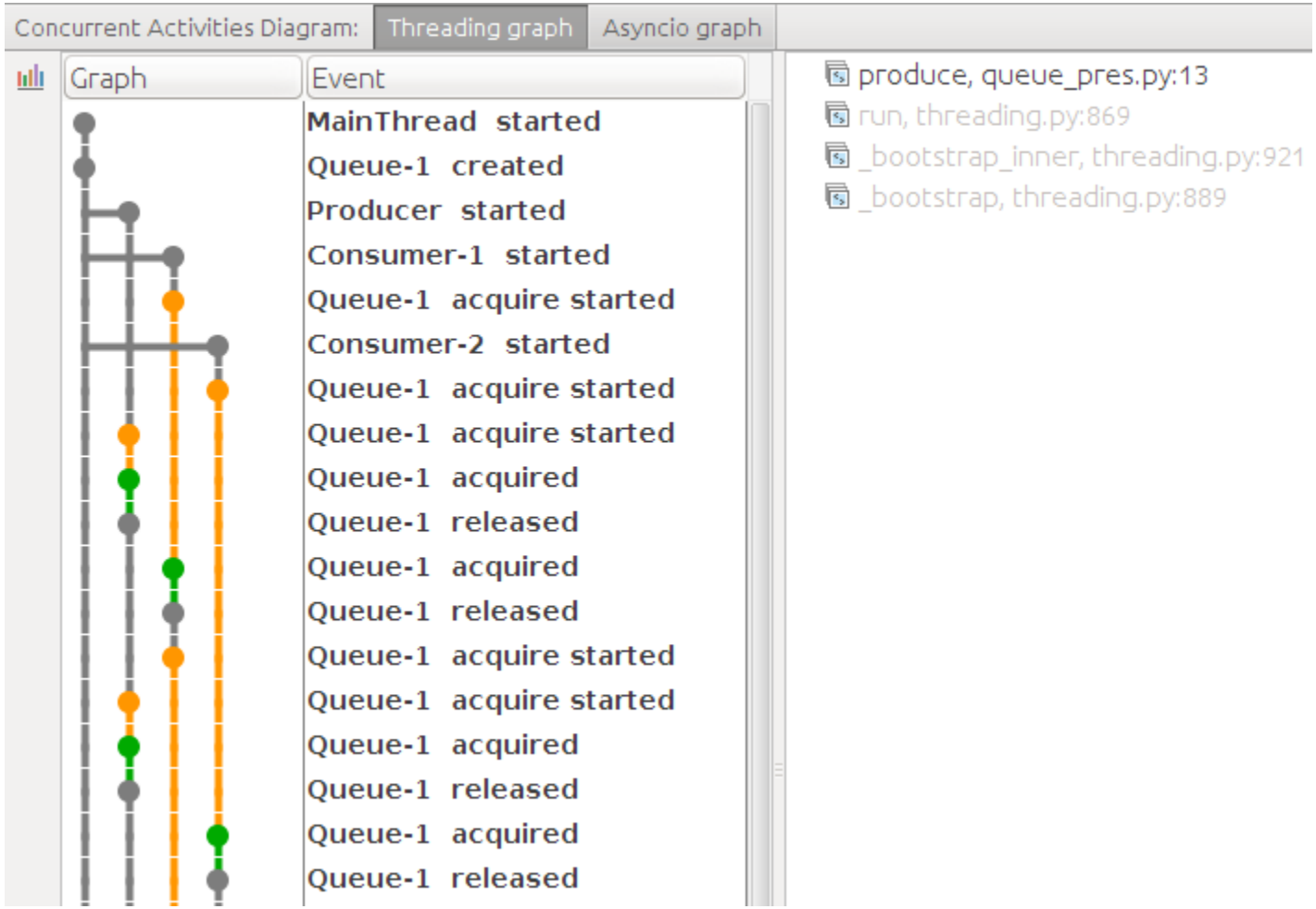
Архитектура системы



Протоколирование событий

- Динамическая модификация кода библиотек
  - threading, asyncio
- Задачи
  - `Thread`, `Task`
- Примитивы синхронизации
  - `Lock`, `RLock`, `Queue`

Пример работы программы



Дефекты в многозадачных программах

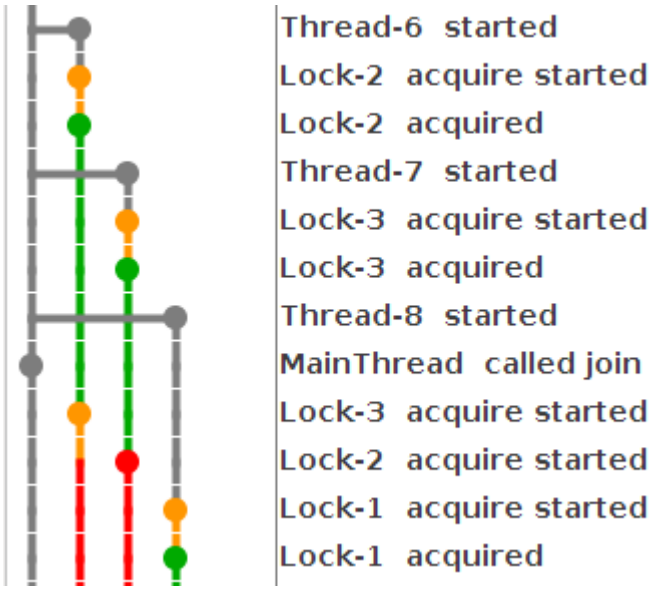
- Неиспользованные объекты синхронизации
- Потоки, не дождавшиеся объединения
- Взаимные блокировки (deadlock)

Взаимные блокировки

```
def f(lock1, lock2):
    lock1.acquire()
    # do work
    lock2.acquire()
    # do another work
    lock2.release()
    lock1.release()

Thread(target=f, name="Thread-1",
       args=(first, second)).start()
Thread(target=f, name="Thread-2",
       args=(second, first)).start()
```

Поиск взаимных блокировок



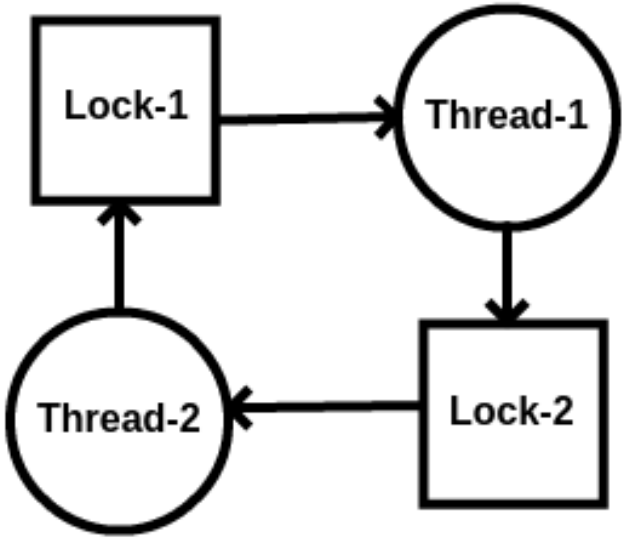


Рис. 4. Пример работы программы с возникновением взаимной блокировки.

Тестирование

- Функциональное тестирование
- Реальные проекты
  - Веб-фреймворк Oscar для Django
  - aiohttp - HTTP client/server
- Эксперимент с участием бета-тестировщиков

Результаты

- Реализовано протоколирование событий в многозадачной программе на языке Python, запущенной в режиме отладки в PyCharm
- Реализовано расширение к PyCharm, осуществляющее:
  - визуализацию исполнения многозадачной программы
  - навигацию к исходному коду и отображение стека вызовов в момент наступления события
- Реализован поиск дефектов в многозадачных программах
  - неиспользованные объекты синхронизации
  - потоки, не дождавшиеся объединения
  - взаимные блокировки (deadlock)
- Проведено тестирование полученного инструмента
- Планируется использование в PyCharm