**BC/NW 2024 № 1 (41):12.2**

**ОБЗОР МЕТОДОВ СТАТИЧЕСКОГО АНАЛИЗА КОДА В PYTHON ПРИ ОТКЛЮЧЕННОЙ ГЛОБАЛЬНОЙ БЛОКИРОВКЕ ИНТЕРПРЕТАТОРА**

Зайнутдинов М.М , Д.Г. Скоробогатов Д.Г., Хиль С.Ю., Орлов Д.А.

Многопоточность играет ключевую роль в современной разработке программного обеспечения, обеспечивая возможность параллельного выполнения задач и повышение производительности приложений. Многопоточное программирование становится особенно важным, поскольку современные процессоры являются многоядерными.

Однако в Python существует ограничение в виде глобальной блокировки интерпретатора (*GIL*). *GIL* является механизмом обеспечения потокобезопасности, но также становится преградой для эффективного использования многопоточности. В Python, из-за глобальной блокировки интерпретатора, одномоментно может выполняться только один поток, что ограничивает потенциал многоядерной обработки.

Для обеспечения высокой эффективности параллельного выполнения можно рассмотреть отключение *GIL*. Однако, отключение *GIL* может привести к возникновению ошибок в программе, таких как взаимная блокировка и эффект гонок [1].

Решением вышеописанных проблем является использование статических анализаторов. Анализаторы могут помочь обнаруживать ошибки и оптимизировать код для эффективного использования многопоточности. В рамках данной работы были рассмотрены четыре варианта программ, представляющих сценарии с взаимной блокировкой, включая ожидание процесса самого себя, взаимное ожидание друг друга, неосвобождение ресурса и получение блокировок в разных порядках, с целью анализа их обнаружения существующими статическими анализаторами в *Python* в контексте работы при отключенной *GIL*.

В рамках исследования использовались следующие статические анализаторы кода: *mypy* [2], *pylint*, *flake8* и *bandit*.

По результатам анализа программных вариантов с взаимной блокировкой установлено, что ни один из использованных статических анализаторов не выявил ошибки взаимной блокировки. Таким образом, задача разработки инструментов для статического анализа программ, ориентированных на обнаружение ошибок, связанных с многопоточностью, является актуальной.

**Литература**

1. **Sam Gross.** Making the Global Interpreter Lock Optional in CPython. Python Enhancement Proposals. [Электронный ресурс] URL: <https://peps.python.org/pep-0703/>
2. 2. **Jukka Lehtosalo.** Mypy 1.7.0 documentation. [Электронный ресурс] URL: <https://mypy.readthedocs.io/en/stable/>