
Suporte experimental do Python para threads livres

Release 3.14.0a7

Guido van Rossum and the Python development team

abril 27, 2025

Python Software Foundation
Email: docs@python.org


Sumário

1	Instalação	2
2	Identificando um Python com threads livres	2
3	A trava global do interpretador no Python com threads livres	2
4	Segurança nas threads	2
5	Limitações conhecidas	3
5.1	Imortalização	3
5.2	Objetos quadro	3
5.3	Iteradores	3
5.4	Desempenho com thread única	3
6	Behavioral changes	3
6.1	Context variables	3
6.2	Warning filters	4
	Índice	5

A partir da versão 3.13, o CPython tem suporte experimental para uma compilação do Python chamada threads livres (em inglês, free threading), onde a trava global do interpretador (GIL) está desabilitada. A execução com threads livres permite a utilização total do poder de processamento disponível, executando threads em paralelo nos núcleos de CPU disponíveis. Embora nem todos os softwares se beneficiem disso automaticamente, os programas projetados com uso de threads em mente serão executados mais rapidamente em hardware com vários núcleos.

O modo de threads livres é experimental e há um trabalho contínuo para melhorá-lo: espere alguns bugs e uma perda substancial no desempenho da thread única.

Este documento descreve as implicações de threads livres para o código Python. Veja [freethreading-extensions-howto](#) para informações sobre como escrever extensões C que oferecem suporte à construção com threads livres.

 [Ver também](#)

1 Instalação

A partir do Python 3.13, os instaladores oficiais do macOS e do Windows oferecem suporte opcional à instalação de binários Python com threads livres. Os instaladores estão disponíveis em <https://www.python.org/downloads/>.

Para obter informações sobre outras plataformas, consulte [Instalando um Python com threads livres](#), um guia de instalação mantido pela comunidade para instalar o Python com threads livres

Ao construir o CPython a partir do código-fonte, a opção de configuração `--disable-gil` deve ser usada para construir um interpretador Python com threads livres.

2 Identificando um Python com threads livres

Para verificar se o interpretador atual oferece suporte a threads livres, `python -VV` e `sys.version` contêm “experimental free-threading build”. A nova função `sys._is_gil_enabled()` pode ser usada para verificar se a GIL está realmente desabilitada no processo em execução.

A variável de configuração `sysconfig.get_config_var("Py_GIL_DISABLED")` pode ser usada para determinar se a compilação tem suporte a threads livres. Se a variável for definida como 1, a compilação tem suporte a threads livres. Este é o mecanismo recomendado para decisões relacionadas à configuração da construção.

3 A trava global do interpretador no Python com threads livres

Construções com threads livres do CPython oferecem suporte a opcionalmente executar com a GIL habilitada em tempo de execução usando a variável de ambiente `PYTHON_GIL` ou a opção de linha de comando `-X gil`.

A GIL também pode ser habilitada automaticamente ao importar um módulo de extensão da API C que não esteja explicitamente marcado como oferecendo suporte a threads livres. Um aviso será impresso neste caso.

Além da documentação de pacotes individuais, os seguintes sites rastreiam o status do suporte de pacotes populares para threads livres:

- <https://py-free-threading.github.io/tracking/>
- <https://hugovk.github.io/free-threaded-wheels/>

4 Segurança nas threads

A construção com threads livres do CPython visa fornecer comportamento de segurança às threads semelhante no nível do Python à construção padrão com GIL habilitada. Tipos embutidos como `dict`, `list` e `set` usam travas internas para proteger contra modificações simultâneas de maneiras que se comportam de forma semelhante à GIL. No entanto, o Python não garantiu historicamente um comportamento específico para modificações simultâneas a esses tipos embutidos, portanto, isso deve ser tratado como uma descrição da implementação atual, não uma garantia de comportamento atual ou futuro.

Nota

É recomendável usar `threading.Lock` ou outras primitivas de sincronização em vez de depender de travas internas de tipos embutidos, quando possível.

5 Limitações conhecidas

Esta seção descreve as limitações conhecidas da construção do Python com threads livres.

5.1 Imortalização

A construção com threads livres da versão 3.13 torna alguns objetos imortais. Objetos imortais não são desalocados e têm contagens de referência que nunca são modificadas. Isso é feito para evitar contenção de contagem de referências que impediria o dimensionamento multithread eficiente.

Um objeto será tornado imortal quando um novo thread for iniciado pela primeira vez após o thread principal estar em execução. Os seguintes objetos são imortalizados:

- objetos função declarados no nível de módulo
- descritores de métodos
- objetos código
- objetos módulo e seus dicionários
- classes (objetos de tipo)

Como objetos imortais nunca são desalocados, aplicações que criam muitos objetos desses tipos podem ver aumento no uso de memória. Espera-se que isso seja resolvido na versão 3.14.

Além disso, literais numéricos e de string no código, bem como strings retornadas por `sys.intern()` também são imortalizados. Espera-se que esse comportamento permaneça na construção com threads livres do 3.14.

5.2 Objetos quadro

Não é seguro acessar objetos quadro de outros threads e fazer isso pode fazer com que seu programa trave. Isso significa que `sys._current_frames()` geralmente não é seguro para uso em uma construção com threads livres. Funções como `inspect.currentframe()` e `sys._getframe()` são geralmente seguras, desde que o objeto quadro resultante não seja passado para outro thread.

5.3 Iteradores

Compartilhar o mesmo objeto iterador entre vários threads geralmente não é seguro e os threads podem ver elementos duplicados ou ausentes ao iterar ou travar o interpretador.

5.4 Desempenho com thread única

A construção com threads livres tem sobrecarga adicional ao executar código Python em comparação com a construção padrão com GIL habilitada. Na versão 3.13, essa sobrecarga é de cerca de 40% no conjunto de ferramentas [pyperformance](#). Programas que passam a maior parte do tempo em extensões C ou E/S verão menos impacto. O maior impacto é porque o interpretador adaptativo especializado ([PEP 659](#)) está desabilitado na construção com threads livres. Esperamos reabilitá-lo de forma segura para threads na versão 3.14. Espera-se que essa sobrecarga seja reduzida na próxima versão do Python. Estamos buscando uma sobrecarga de 10% ou menos no conjunto de ferramentas [pyperformance](#) em comparação com a construção padrão com GIL habilitada.

6 Behavioral changes

This section describes CPython behavioural changes with the free-threaded build.

6.1 Context variables

In the free-threaded build, the flag `thread_inherit_context` is set to true by default which causes threads created with `threading.Thread` to start with a copy of the `Context()` of the caller of `start()`. In the default GIL-enabled build, the flag defaults to false so threads start with an empty `Context()`.

6.2 Warning filters

In the free-threaded build, the flag `context_aware_warnings` is set to `true` by default. In the default GIL-enabled build, the flag defaults to `false`. If the flag is `true` then the `warnings.catch_warnings` context manager uses a context variable for warning filters. If the flag is `false` then `catch_warnings` modifies the global filters list, which is not thread-safe. See the `warnings` module for more details.

Índice

P

Propostas de Melhorias do Python

PEP 659, [3](#)

PEP 703, [2](#)

PYTHON_GIL, [2](#)

V

variável de ambiente

PYTHON_GIL, [2](#)