

---

# Suporte do Python ao perfilador perf do Linux

Release 3.12.7

Guido van Rossum and the Python development team

outubro 18, 2024

Python Software Foundation  
Email: docs@python.org

## Sumário

1	Como habilitar o suporte a perfilação com <code>perf</code>	4
2	Como obter os melhores resultados	4
	Índice	6

---

### autor

Pablo Galindo

O perfilador `perf` do Linux é uma ferramenta muito poderosa que permite criar perfis e obter informações sobre o desempenho da sua aplicação. `perf` também possui um ecossistema muito vibrante de ferramentas que auxiliam na análise dos dados que produz.

O principal problema de usar o perfilador `perf` com aplicações Python é que `perf` apenas obtém informações sobre símbolos nativos, ou seja, os nomes de funções e procedimentos escritos em C. Isso significa que os nomes de funções Python e seus nomes de arquivos em seu código não aparecerão na saída de `perf`.

Desde o Python 3.12, o interpretador pode ser executado em um modo especial que permite que funções do Python apareçam na saída do criador de perfilador `perf`. Quando este modo está habilitado, o interpretador interporá um pequeno pedaço de código compilado instantaneamente antes da execução de cada função Python e ensinará `perf` a relação entre este pedaço de código e a função Python associada usando arquivos de mapa `perf`.

### Nota

O suporte para o perfilador `perf` está atualmente disponível apenas para Linux em arquiteturas selecionadas. Verifique a saída da etapa de construção `configure` ou verifique a saída de `python -m sysconfig | grep HAVE_PERF_TRAMPOLINE` para ver se o seu sistema é compatível.

Por exemplo, considere o seguinte script:

```
def foo(n):  
    result = 0  
    for _ in range(n):  
        result += 1
```

(continua na próxima página)

(continuação da página anterior)

```

    return result

def bar(n):
    foo(n)

def baz(n):
    bar(n)

if __name__ == "__main__":
    baz(1000000)

```

Podemos executar `perf` para obter amostras de rastreamentos de pilha da CPU em 9999 hertz:

```
$ perf record -F 9999 -g -o perf.data python my_script.py
```

Então podemos usar `perf report` para analisar os dados:

```

$ perf report --stdio -n -g

# Children      Self          Samples  Command      Shared Object      Symbol
# .....      .....      .....      .....      .....      .....
↪ .....
#
    91.08%      0.00%              0  python.exe  python.exe      [...] _start
    |
    ---_start
    |
        --90.71%--__libc_start_main
            Py_BytesMain
                |
                |--56.88%--pymain_run_python.constprop.0
                |
                |--56.13%--_PyRun_AnyFileObject
                |               _PyRun_SimpleFileObject
                |               |
                |               |--55.02%--run_mod
                |               |
                |               --54.65%--PyEval_EvalCode
                |                       _PyEval_
↪EvalFrameDefault
                |               |
↪Vectorcall
                |               |
                |               _PyEval_Vector
                |               _PyEval_
↪EvalFrameDefault
                |               |
↪Vectorcall
                |               |
                |               _PyEval_Vector
                |               _PyEval_
↪EvalFrameDefault
                |               |
↪Vectorcall
                |               |
                |               _PyEval_Vector
                |               |
                |               |--51.67%--_
↪PyEval_EvalFrameDefault
                |               |
                |               |

```

(continua na próxima página)

(continuação da página anterior)

↪11.52%--_PyLong_Add						--
↪						└
↪						└
↪		--2.97%--_PyObject_Malloc				
...						

Como você pode ver, as funções Python não são mostradas na saída, apenas `_PyEval_EvalFrameDefault` (a função que avalia o bytecode Python) aparece. Infelizmente isso não é muito útil porque todas as funções Python usam a mesma função C para avaliar bytecode, portanto não podemos saber qual função Python corresponde a qual função de avaliação de bytecode.

Em vez disso, se executarmos o mesmo experimento com o suporte `perf` ativado, obteremos:

```
$ perf report --stdio -n -g

# Children      Self          Samples  Command      Shared Object      Symbol
# .....
# .....
#
  90.58%      0.36%              1  python.exe  python.exe        [.] _start
    |
    ---_start
    |
      --89.86%--__libc_start_main
        Py_BytesMain
          |
          |--55.43%--pymain_run_python.constprop.0
          |
          |      |--54.71%--_PyRun_AnyFileObject
          |      |      _PyRun_SimpleFileObject
          |      |
          |      |--53.62%--run_mod
          |      |
          |      |      --53.26%--PyEval_EvalCode
          |      |      py::<module>:/
↪src/script.py
↪EvalFrameDefault
↪Vectorcall
    |
    |      _PyEval_Vector
    |      py::baz:/src/
↪script.py
↪EvalFrameDefault
↪Vectorcall
    |
    |      _PyEval_Vector
    |      py::bar:/src/
↪script.py
↪EvalFrameDefault
↪Vectorcall
    |
    |      _PyEval_Vector
```

(continua na próxima página)

(continuação da página anterior)

				py::foo:/src/
↪script.py				
				--51.81%--_
↪PyEval_EvalFrameDefault				
↪13.77%--_PyObject_Add				
↪				
↪ --3.26%--_PyObject_Malloc				

## 1 Como habilitar o suporte a perfilação com perf

O suporte à perfilação com perf pode ser habilitado desde o início usando a variável de ambiente PYTHONPERFSUPPORT ou a opção -X perf, ou dinamicamente usando sys.activate\_stack\_trampoline() e sys.deactivate\_stack\_trampoline().

As funções `sys` têm precedência sobre a opção `-X`, a opção `-X` tem precedência sobre a variável de ambiente.

Exemplo usando a variável de ambiente:

```
$ PYTHONPERFSUPPORT=1 python script.py
$ perf report -q -i perf.data
```

Exemplo usando a opção -X:

```
$ python -X perf script.py
$ perf report -q -i perf.data
```

Exemplo usando as APIs de `sys` em `example.py`:

```
import sys

sys.activate_stack_trampoline("perf")
do_profiled_stuff()
sys.deactivate_stack_trampoline()

non_profiled_stuff()
```

... então:

```
$ python ./example.py
$ perf report -q -i perf.data
```

## 2 Como obter os melhores resultados

Para melhores resultados, Python deve ser compilado com `CFLAGS="-fno-omit-frame-pointer -mno-omit-leaf-frame-pointer"`, pois isso permite que os perfiladores façam o desenrolamento de pilha (ou *stack unwinding*) usando apenas o ponteiro de quadro e não no DWARF informações de depuração. Isso ocorre porque como o código interposto para permitir o suporte `perf` é gerado dinamicamente, ele não possui nenhuma informação de depuração DWARF disponível.

Você pode verificar se o seu sistema foi compilado com este sinalizador executando:

```
$ python -m sysconfig | grep 'no-omit-frame-pointer'
```

Se você não vir nenhuma saída, significa que seu interpretador não foi compilado com ponteiros de quadro e, portanto, pode não ser capaz de mostrar funções Python na saída de `perf`.

## Índice

### P

`PYTHONPERFSUPPORT`, 4

### V

variável de ambiente  
    `PYTHONPERFSUPPORT`, 4