Depurando extensões de API C e internos do CPython com GDB

Release 3.12.7

Guido van Rossum and the Python development team

outubro 18, 2024

Python Software Foundation
Email: docs@python.org

Sumário

Este documento explica como a extensão GDB do Python, `python-gdb.py`, pode ser usada com o depurador GDB para depurar extensões CPython e o próprio interpretador CPython.

Ao depurar problemas de baixo nível, como falhas ou impasses, um depurador de baixo nível, como o GDB, é útil para diagnosticar e corrigir o problema. Por padrão, o GDB (ou qualquer uma de suas interfaces) não oferece suporte a informações de alto nível específicas do interpretador CPython.

A extensão `python-gdb.py` adiciona informações do interpretador CPython ao GDB. A extensão ajuda a inspecionar a pilha de funções Python atualmente em execução. Dado um objeto Python representado por um ponteiro `PyObject*`, a extensão mostra o tipo e o valor do objeto.

Desenvolvedores que estão trabalhando em extensões CPython ou mexendo com partes do CPython escritas em C podem usar este documento para aprender como usar a extensão `python-gdb.py` com o GDB.

> **Nota**
>
> Este documento pressupõe que você esteja familiarizado com o básico do GDB e da API C do CPython. Ele consolida orientações do devguide e da wiki do Python.

1 Pré-requisitos

Você precisa ter:

- GDB 7 ou posterior. (Para versões anteriores do GDB, consulte `Misc/gdbinit` nas fontes do Python 3.11 ou anterior.)
- Informações de depuração compatíveis com GDB para Python e qualquer extensão que você esteja depurando.
- A extensão `python-gdb.py`.

A extensão é construída com Python, mas pode ser distribuída separadamente ou não ser distribuída. Abaixo, incluímos dicas para alguns sistemas comuns como exemplos. Note que mesmo se as instruções corresponderem ao seu sistema, elas podem estar desatualizadas.

1.1 Configuração com Python construído a partir do código-fonte

Quando você compila o CPython a partir do código-fonte, as informações de depuração devem estar disponíveis, e a compilação deve adicionar um arquivo `python-gdb.py` ao diretório raiz do seu repositório.

Para ativar o suporte, você deve adicionar o diretório que contém `python-gdb.py` ao "auto-load-safe-path" do GDB. Se você ainda não fez isso, as versões recentes do GDB imprimirão um aviso com instruções sobre como fazer isso.

Nota

Se você não encontrar instruções para a sua versão do GDB, coloque isso no seu arquivo de configuração (`~/.gdbinit` ou `~/.config/gdb/gdbinit`):

```
add-auto-load-safe-path /path/to/cpython
```

Você também pode adicionar vários caminhos, separados por `:`.

1.2 Configuração para Python a partir de uma distribuição Linux

A maioria dos sistemas Linux fornece informações de depuração para o Python do sistema em um pacote chamado `python-debuginfo`, `python-dbg` ou similar. Por exemplo:

- Fedora:

```
sudo dnf install gdb
sudo dnf debuginfo-install python3
```

- Ubuntu:

```
sudo apt install gdb python3-dbg
```

Em vários sistemas Linux recentes, o GDB pode baixar automaticamente símbolos de depuração usando *debuginfod*. No entanto, isso não instalará a extensão `python-gdb.py`; geralmente é necessário instalar separadamente o pacote de informações de depuração.

2 Usando a compilação de depuração e o modo de desenvolvimento

Para facilitar a depuração, você pode querer:

- Use a compilação de depuração do Python. (Ao compilar a partir do código-fonte, use `configure --with-pydebug`. Em distribuições Linux, instale e execute um pacote como `python-debug` ou `python-dbg`, se disponível.)
- Use o modo de desenvolvimento de tempo de execução (`-X dev`).

Ambos habilitam assertivas extras e desabilitam algumas otimizações. Às vezes isso esconde o bug que você está tentando encontrar, mas na maioria dos casos eles facilitam o processo.

3 Usando a extensão `python-gdb`

Quando a extensão é carregada, ela fornece duas principais funcionalidades: impressões bonitas para valores Python e comandos adicionais.

3.1 Pretty-printers

Este é o aspecto de um backtrace do GDB (truncado) quando esta extensão está habilitada:

```
#0  0x000000000041a6b1 in PyObject_Malloc (nbytes=Cannot access memory at address␣
↪0x7fffff7fefe8
) at Objects/obmalloc.c:748
#1  0x000000000041b7c0 in _PyObject_DebugMallocApi (id=111 'o', nbytes=24) at␣
↪Objects/obmalloc.c:1445
#2  0x000000000041b717 in _PyObject_DebugMalloc (nbytes=24) at Objects/obmalloc.
↪c:1412
#3  0x000000000044060a in _PyUnicode_New (length=11) at Objects/unicodeobject.c:346
#4  0x00000000004466aa in PyUnicodeUCS2_DecodeUTF8Stateful (s=0x5c2b8d "__lltrace__
↪", size=11, errors=0x0, consumed=
    0x0) at Objects/unicodeobject.c:2531
#5  0x0000000000446647 in PyUnicodeUCS2_DecodeUTF8 (s=0x5c2b8d "__lltrace__",␣
↪size=11, errors=0x0)
    at Objects/unicodeobject.c:2495
#6  0x0000000000440d1b in PyUnicodeUCS2_FromStringAndSize (u=0x5c2b8d "__lltrace__
↪", size=11)
    at Objects/unicodeobject.c:551
#7  0x0000000000440d94 in PyUnicodeUCS2_FromString (u=0x5c2b8d "__lltrace__") at␣
↪Objects/unicodeobject.c:569
#8  0x0000000000584abd in PyDict_GetItemString (v=
    {'Yuck': <type at remote 0xad4730>, '__builtins__': <module at remote␣
↪0x7ffff7fd5ee8>, '__file__': 'Lib/test/crashers/nasty_eq_vs_dict.py', '__package_
↪_': None, 'y': <Yuck(i=0) at remote 0xaacd80>, 'dict': {0: 0, 1: 1, 2: 2, 3: 3},
↪'__cached__': None, '__name__': '__main__', 'z': <Yuck(i=0) at remote 0xaace60>,
↪'__doc__': None}, key=
    0x5c2b8d "__lltrace__") at Objects/dictobject.c:2171
```

Observe como o argumento do dicionário para `PyDict_GetItemString` é exibido como seu `repr()`, em vez de um ponteiro `PyObject *` opaco.

A extensão funciona fornecendo uma rotina de impressão personalizada para valores do tipo `PyObject *`. Se você precisar acessar detalhes de nível inferior de um objeto, então converta o valor para um ponteiro do tipo apropriado. Por exemplo:

```
(gdb) p globals
$1 = {'__builtins__': <module at remote 0x7ffff7fb1868>, '__name__':
'__main__', 'ctypes': <module at remote 0x7ffff7f14360>, '__doc__': None,
'__package__': None}

(gdb) p *(PyDictObject*)globals
$2 = {ob_refcnt = 3, ob_type = 0x3dbdf85820, ma_fill = 5, ma_used = 5,
ma_mask = 7, ma_table = 0x63d0f8, ma_lookup = 0x3dbdc7ea70
<lookdict_string>, ma_smalltable = {{me_hash = 7065186196740147912,
me_key = '__builtins__', me_value = <module at remote 0x7ffff7fb1868>},
{me_hash = -368181376027291943, me_key = '__name__',
me_value ='__main__'}, {me_hash = 0, me_key = 0x0, me_value = 0x0},
```

(continua na próxima página)

(continuação da página anterior)

```
{me_hash = 0, me_key = 0x0, me_value = 0x0},
{me_hash = -9177857982131165996, me_key = 'ctypes',
me_value = <module at remote 0x7ffff7f14360>},
{me_hash = -8518757509529533123, me_key = '__doc__', me_value = None},
{me_hash = 0, me_key = 0x0, me_value = 0x0}, {
  me_hash = 6614918939584953775, me_key = '__package__', me_value = None}}}
```

Observe que os pretty-printers não chamam realmente `repr()`. Para tipos básicos, eles tentam corresponder ao seu resultado de perto.

Uma área que pode ser confusa é que a impressão personalizada para alguns tipos se parece muito com a impressão embutida do GDB para tipos padrão. Por exemplo, a impressora bonita para um `int` do Python (`PyLongObject*`) fornece uma representação que não é distinguível de um inteiro de nível de máquina regular.

```
(gdb) p some_machine_integer
$3 = 42

(gdb) p some_python_integer
$4 = 42
```

A estrutura interna pode ser revelada com um elenco para `PyLongObject*`.

> (gdb) p *(PyLongObject)*some_python_integer $5 = {ob_base = {ob_base = {ob_refcnt = 8, ob_type = 0x3dad39f5e0}, ob_size = 1}, ob_digit = {42}}

Uma confusão semelhante pode surgir com o tipo `str`, onde a saída se parece muito com a a impressão embutida do gdb para `char *`.

```
(gdb) p ptr_to_python_str
$6 = '__builtins__'
```

O pretty-printer para instâncias de `str` tem como padrão o uso de aspas simples (assim como o `repr` do Python para strings), enquanto o printer padrão para valores de `char *` usa aspas duplas e contém um endereço hexadecimal.

```
(gdb) p ptr_to_char_star
$7 = 0x6d72c0 "hello world"
```

Novamente, os detalhes de implementação podem ser revelados com um chamada a `PyUnicodeObject*`::.

```
(gdb) p *(PyUnicodeObject*)$6
$8 = {ob_base = {ob_refcnt = 33, ob_type = 0x3dad3a95a0}, length = 12,
str = 0x7ffff2128500, hash = 7065186196740147912, state = 1, defenc = 0x0}
```

3.2 py-list

A extensão adiciona um comando `py-list`, que lista o código-fonte Python (se houver) para o quadro atual na thread selecionada. A linha atual é marcada com um “>”:

```
(gdb) py-list
 901        if options.profile:
 902            options.profile = False
 903            profile_me()
 904            return
 905
>906        u = UI()
 907        if not u.quit:
 908            try:
 909                gtk.main()
```

(continua na próxima página)

(continuação da página anterior)

```
910             except KeyboardInterrupt:
911                 # properly quit on a keyboard interrupt...
```

Use `py-list START` para listar em um número de linha diferente dentro do código Python, e `py-list START,END` para listar um intervalo específico de linhas dentro do código Python.

3.3 py-up e py-down

Os comandos `py-up` e `py-down` são análogos aos comandos regulares `up` e `down` do GDB, mas tentam se mover no nível dos quadros do CPython, em vez dos quadros do C.

GDB nem sempre consegue ler as informações relevantes do quadro, dependendo do nível de otimização com o qual o CPython foi compilado. Internamente, os comandos procuram por quadros C que estão executando a função de avaliação de quadro padrão (ou seja, o laço do interpretador de bytecode principal dentro do CPython) e procuram o valor do `PyFrameObject *` relacionado.

Eles emitem o número do quadro (no nível C) dentro da thread.

Por exemplo:

```
(gdb) py-up
#37 Frame 0x9420b04, for file /usr/lib/python2.6/site-packages/
gnome_sudoku/main.py, line 906, in start_game ()
    u = UI()
(gdb) py-up
#40 Frame 0x948e82c, for file /usr/lib/python2.6/site-packages/
gnome_sudoku/gnome_sudoku.py, line 22, in start_game(main=<module at␣
↪remote 0xb771b7f4>)
    main.start_game()
(gdb) py-up
Unable to find an older python frame
```

de forma estamos no topo da pilha do Python.

Os números de quadro correspondem aos exibidos pelo comando padrão `backtrace` do GDB. O comando ignora os quadros C que não estão executando código Python.

Voltando para baixo:

```
(gdb) py-down
#37 Frame 0x9420b04, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/main.py, line 906, in start_game ()
    u = UI()
(gdb) py-down
#34 (unable to read python frame information)
(gdb) py-down
#23 (unable to read python frame information)
(gdb) py-down
#19 (unable to read python frame information)
(gdb) py-down
#14 Frame 0x99262ac, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/game_selector.py, line 201, in run_swallowed_dialog (self=
↪<NewOrSavedGameSelector(new_game_model=<gtk.ListStore at remote␣
↪0x98fab44>, puzzle=None, saved_games=[{'gsd.auto_fills': 0, 'tracking':
↪{}, 'trackers': {}, 'notes': [], 'saved_at': 1270084485, 'game': '7 8 0␣
↪0 0 0 0 5 6 0 0 9 0 8 0 1 0 0 0 4 6 0 0 0 0 7 0 6 5 0 0 0 4 7 9 2 0 0 0␣
↪9 0 1 0 0 0 3 9 7 6 0 0 0 1 8 0 6 0 0 0 0 2 8 0 0 0 5 0 4 0 6 0 0 2 1 0␣
↪0 0 0 0 4 5\n7 8 0 0 0 0 0 5 6 0 0 9 0 8 0 1 0 0 0 4 6 0 0 0 0 7 0 6 5␣
↪1 8 3 4 7 9 2 0 0 0 9 0 1 0 0 0 3 9 7 6 0 0 0 1 8 0 6 0 0 0 0 2 8 0 0 0␣
```

(continua na próxima página)

(continuação da página anterior)

```
↪5 0 4 0 6 0 0 2 1 0 0 0 0 0 4 5', 'gsd.impossible_hints': 0, 'timer.__
↪absolute_start_time__': <float at remote 0x984b474>, 'gsd.hints': 0,
↪'timer.active_time': <float at remote 0x984b494>, 'timer.total_time':
↪<float at remote 0x984b464>}], dialog=<gtk.Dialog at remote 0x98faaa4>,␣
↪saved_game_model=<gtk.ListStore at remote 0x98fad24>, sudoku_maker=
↪<SudokuMaker(terminated=False, played=[], batch_siz...(truncated)
            swallower.run_dialog(self.dialog)
(gdb) py-down
#11 Frame 0x9aead74, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/dialog_swallower.py, line 48, in run_dialog (self=
↪<SwappableArea(running=<gtk.Dialog at remote 0x98faaa4>, main_page=0)␣
↪at remote 0x98fa6e4>, d=<gtk.Dialog at remote 0x98faaa4>)
            gtk.main()
(gdb) py-down
#8 (unable to read python frame information)
(gdb) py-down
Unable to find a newer python frame
```

e estamos na parte inferior da pilha do Python.

Observe que no Python 3.12 e versões mais recentes, o mesmo quadro de pilha C pode ser usado para vários quadros de pilha Python. Isso significa que `py-up` e `py-down` podem mover vários quadros Python de uma vez. Por exemplo:

```
(gdb) py-up
#6 Frame 0x7ffff7fb62b0, for file /tmp/rec.py, line 5, in recursive_
↪function (n=0)
   time.sleep(5)
#6 Frame 0x7ffff7fb6240, for file /tmp/rec.py, line 7, in recursive_
↪function (n=1)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb61d0, for file /tmp/rec.py, line 7, in recursive_
↪function (n=2)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6160, for file /tmp/rec.py, line 7, in recursive_
↪function (n=3)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb60f0, for file /tmp/rec.py, line 7, in recursive_
↪function (n=4)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6080, for file /tmp/rec.py, line 7, in recursive_
↪function (n=5)
   recursive_function(n-1)
#6 Frame 0x7ffff7fb6020, for file /tmp/rec.py, line 9, in <module> ()
   recursive_function(5)
(gdb) py-up
Unable to find an older python frame
```

3.4 py-bt

O comando `py-bt` tenta mostrar uma rastreabilidade em nível Python da thread atual.

Por exemplo:

```
(gdb) py-bt
#8 (unable to read python frame information)
#11 Frame 0x9aead74, for file /usr/lib/python2.6/site-packages/gnome_
```

(continua na próxima página)

(continuação da página anterior)

```
↪sudoku/dialog_swallower.py, line 48, in run_dialog (self=
↪<SwappableArea(running=<gtk.Dialog at remote 0x98faaa4>, main_page=0)␣
↪at remote 0x98fa6e4>, d=<gtk.Dialog at remote 0x98faaa4>)
            gtk.main()
#14 Frame 0x99262ac, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/game_selector.py, line 201, in run_swallowed_dialog (self=
↪<NewOrSavedGameSelector(new_game_model=<gtk.ListStore at remote␣
↪0x98fab44>, puzzle=None, saved_games=[{'gsd.auto_fills': 0, 'tracking':
↪{}, 'trackers': {}, 'notes': [], 'saved_at': 1270084485, 'game': '7 8 0␣
↪0 0 0 0 5 6 0 0 9 0 8 0 1 0 0 0 4 6 0 0 0 0 7 0 6 5 0 0 0 4 7 9 2 0 0 0␣
↪9 0 1 0 0 0 3 9 7 6 0 0 0 1 8 0 6 0 0 0 0 2 8 0 0 0 5 0 4 0 6 0 0 2 1 0␣
↪0 0 0 0 4 5\n7 8 0 0 0 0 0 5 6 0 0 9 0 8 0 1 0 0 0 4 6 0 0 0 0 7 0 6 5␣
↪1 8 3 4 7 9 2 0 0 0 9 0 1 0 0 0 3 9 7 6 0 0 0 1 8 0 6 0 0 0 0 2 8 0 0 0␣
↪5 0 4 0 6 0 0 2 1 0 0 0 0 0 4 5', 'gsd.impossible_hints': 0, 'timer.__
↪absolute_start_time__': <float at remote 0x984b474>, 'gsd.hints': 0,
↪'timer.active_time': <float at remote 0x984b494>, 'timer.total_time':
↪<float at remote 0x984b464>}], dialog=<gtk.Dialog at remote 0x98faaa4>,␣
↪saved_game_model=<gtk.ListStore at remote 0x98fad24>, sudoku_maker=
↪<SudokuMaker(terminated=False, played=[], batch_siz...(truncated)
            swallower.run_dialog(self.dialog)
#19 (unable to read python frame information)
#23 (unable to read python frame information)
#34 (unable to read python frame information)
#37 Frame 0x9420b04, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/main.py, line 906, in start_game ()
    u = UI()
#40 Frame 0x948e82c, for file /usr/lib/python2.6/site-packages/gnome_
↪sudoku/gnome_sudoku.py, line 22, in start_game (main=<module at remote␣
↪0xb771b7f4>)
    main.start_game()
```

Os números de quadro correspondem aos exibidos pelo comando padrão `backtrace` do GDB.

3.5 py-print

O comando `py-print` procura um nome em Python e tenta imprimi-lo. Ele procura em locais dentro da thread atual, depois em globais e, finalmente, em embutidos:

```
(gdb) py-print self
local 'self' = <SwappableArea(running=<gtk.Dialog at remote 0x98faaa4>,
main_page=0) at remote 0x98fa6e4>
(gdb) py-print __name__
global '__name__' = 'gnome_sudoku.dialog_swallower'
(gdb) py-print len
builtin 'len' = <built-in function len>
(gdb) py-print scarlet_pimpernel
'scarlet_pimpernel' not found
```

Se o quadro C atual corresponder a vários quadros Python, `py-print` considera apenas o primeiro.

3.6 py-locals

O comando `py-locals` busca todas as variáveis locais do Python no quadro Python atual na thread selecionada e imprime suas representações:

```
(gdb) py-locals
self = <SwappableArea(running=<gtk.Dialog at remote 0x98faaa4>,
```

(continua na próxima página)

(continuação da página anterior)

```
main_page=0) at remote 0x98fa6e4>
d = <gtk.Dialog at remote 0x98faaa4>
```

Se o quadro C atual corresponder a vários quadros Python, serão mostrados os locais de todos eles:

```
(gdb) py-locals
Locals for recursive_function
n = 0
Locals for recursive_function
n = 1
Locals for recursive_function
n = 2
Locals for recursive_function
n = 3
Locals for recursive_function
n = 4
Locals for recursive_function
n = 5
Locals for <module>
```

4 Uso com comandos do GDB

Os comandos de extensão complementam os comandos embutidos do GDB. Por exemplo, você pode usar os números de quadro mostrados por `py-bt` com o comando `frame` para ir a um quadro específico dentro da thread selecionada, assim:

```
(gdb) py-bt
(output snipped)
#68 Frame 0xaa4560, for file Lib/test/regrtest.py, line 1548, in <module> ()
        main()
(gdb) frame 68
#68 0x00000000004cd1e6 in PyEval_EvalFrameEx (f=Frame 0xaa4560, for file Lib/test/
↪regrtest.py, line 1548, in <module> (), throwflag=0) at Python/ceval.c:2665
2665                            x = call_function(&sp, oparg);
(gdb) py-list
1543        # Run the tests in a context manager that temporary changes the CWD to␣
↪a
1544        # temporary and writable directory. If it's not possible to create or
1545        # change the CWD, the original CWD will be used. The original CWD is
1546        # available from test_support.SAVEDCWD.
1547        with test_support.temp_cwd(TESTCWD, quiet=True):
>1548            main()
```

O comando `info threads` fornecerá uma lista das threads dentro do processo, e você pode usar o comando `thread` para selecionar uma diferente:

```
(gdb) info threads
  105 Thread 0x7fffefa18710 (LWP 10260)  sem_wait () at ../nptl/sysdeps/unix/sysv/
↪linux/x86_64/sem_wait.S:86
  104 Thread 0x7fffdf5fe710 (LWP 10259)  sem_wait () at ../nptl/sysdeps/unix/sysv/
↪linux/x86_64/sem_wait.S:86
* 1 Thread 0x7ffff7fe2700 (LWP 10145)  0x00000038e46d73e3 in select () at ../
↪sysdeps/unix/syscall-template.S:82
```

Você pode usar `thread apply all COMANDO` ou (`t a a COMANDO` para abreviar) para executar um comando em todas as threads. Com `py-bt`, isso permite que você veja o que cada thread está fazendo no nível do Python:

```
(gdb) t a a py-bt

Thread 105 (Thread 0x7fffefa18710 (LWP 10260)):
#5 Frame 0x7fffd00019d0, for file /home/david/coding/python-svn/Lib/threading.py,␣
↪line 155, in _acquire_restore (self=<_RLock(_Verbose__verbose=False, _RLock__
↪owner=140737354016512, _RLock__block=<thread.lock at remote 0x858770>, _RLock__
↪count=1) at remote 0xd7ff40>, count_owner=(1, 140737213728528), count=1,␣
↪owner=140737213728528)
        self.__block.acquire()
#8 Frame 0x7fffac001640, for file /home/david/coding/python-svn/Lib/threading.py,␣
↪line 269, in wait (self=<_Condition(_Condition__lock=<_RLock(_Verbose__
↪verbose=False, _RLock__owner=140737354016512, _RLock__block=<thread.lock at␣
↪remote 0x858770>, _RLock__count=1) at remote 0xd7ff40>, acquire=<instancemethod␣
↪at remote 0xd80260>, _is_owned=<instancemethod at remote 0xd80160>, _release_
↪save=<instancemethod at remote 0xd803e0>, release=<instancemethod at remote␣
↪0xd802e0>, _acquire_restore=<instancemethod at remote 0xd7ee60>, _Verbose__
↪verbose=False, _Condition__waiters=[]) at remote 0xd7fd10>, timeout=None, waiter=
↪<thread.lock at remote 0x858a90>, saved_state=(1, 140737213728528))
            self._acquire_restore(saved_state)
#12 Frame 0x7fffb8001a10, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 348, in f ()
            cond.wait()
#16 Frame 0x7fffb8001c40, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 37, in task (tid=140737213728528)
                f()

Thread 104 (Thread 0x7fffdf5fe710 (LWP 10259)):
#5 Frame 0x7fffe4001580, for file /home/david/coding/python-svn/Lib/threading.py,␣
↪line 155, in _acquire_restore (self=<_RLock(_Verbose__verbose=False, _RLock__
↪owner=140737354016512, _RLock__block=<thread.lock at remote 0x858770>, _RLock__
↪count=1) at remote 0xd7ff40>, count_owner=(1, 140736940992272), count=1,␣
↪owner=140736940992272)
        self.__block.acquire()
#8 Frame 0x7fffc8002090, for file /home/david/coding/python-svn/Lib/threading.py,␣
↪line 269, in wait (self=<_Condition(_Condition__lock=<_RLock(_Verbose__
↪verbose=False, _RLock__owner=140737354016512, _RLock__block=<thread.lock at␣
↪remote 0x858770>, _RLock__count=1) at remote 0xd7ff40>, acquire=<instancemethod␣
↪at remote 0xd80260>, _is_owned=<instancemethod at remote 0xd80160>, _release_
↪save=<instancemethod at remote 0xd803e0>, release=<instancemethod at remote␣
↪0xd802e0>, _acquire_restore=<instancemethod at remote 0xd7ee60>, _Verbose__
↪verbose=False, _Condition__waiters=[]) at remote 0xd7fd10>, timeout=None, waiter=
↪<thread.lock at remote 0x858860>, saved_state=(1, 140736940992272))
            self._acquire_restore(saved_state)
#12 Frame 0x7fffac001c90, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 348, in f ()
            cond.wait()
#16 Frame 0x7fffac0011c0, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 37, in task (tid=140736940992272)
                f()

Thread 1 (Thread 0x7ffff7fe2700 (LWP 10145)):
#5 Frame 0xcb5380, for file /home/david/coding/python-svn/Lib/test/lock_tests.py,␣
↪line 16, in _wait ()
    time.sleep(0.01)
#8 Frame 0x7fffd00024a0, for file /home/david/coding/python-svn/Lib/test/lock_
↪tests.py, line 378, in _check_notify (self=<ConditionTests(_testMethodName='test_
↪notify', _resultForDoCleanups=<TestResult(_original_stdout=<cStringIO.StringO at␣
```

(continua na próxima página)

(continuação da página anterior)

```
↪remote 0xc191e0>, skipped=[], _mirrorOutput=False, testsRun=39, buffer=False, _
↪original_stderr=<file at remote 0x7ffff7fc6340>, _stdout_buffer=<cStringIO.
↪StringO at remote 0xc9c7f8>, _stderr_buffer=<cStringIO.StringO at remote␣
↪0xc9c790>, _moduleSetUpFailed=False, expectedFailures=[], errors=[], _
↪previousTestClass=<type at remote 0x928310>, unexpectedSuccesses=[], failures=[],
↪ shouldStop=False, failfast=False) at remote 0xc185a0>, _threads=(0,), _
↪cleanups=[], _type_equality_funcs={<type at remote 0x7eba00>: <instancemethod at␣
↪remote 0xd750e0>, <type at remote 0x7e7820>: <instancemethod at remote 0xd75160>,
↪ <type at remote 0x7e30e0>: <instancemethod at remote 0xd75060>, <type at remote␣
↪0x7e7d20>: <instancemethod at remote 0xd751e0>, <type at remote 0x7f19e0...
↪(truncated)
       _wait()
```