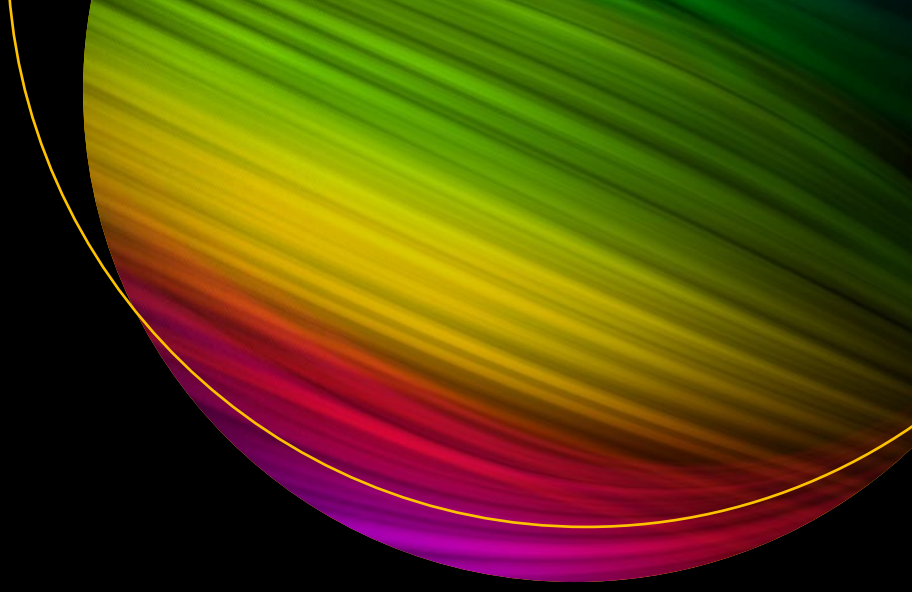

Python Debugging for AI, Machine Learning, and Cloud Computing

A Pattern-Oriented Approach

Dmitry Vostokov

Apress®

Python Debugging for AI, Machine Learning, and Cloud Computing

A Pattern-Oriented Approach

Dmitry Vostokov

Apress®

Python Debugging for AI, Machine Learning, and Cloud Computing: A Pattern-Oriented Approach

Dmitry Vostokov
Dalkey, Dublin, Ireland

ISBN-13 (pbk): 978-1-4842-9744-5 ISBN-13 (electronic): 978-1-4842-9745-2
https://doi.org/10.1007/978-1-4842-9745-2

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: James Markham
Editorial Assistant: Gryffin Winkler
Copy Editor: Mary Behr

Cover designed by eStudioCalamar

Cover image designed by Igor Mamaev on pixabay

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit https://www.apress.com/gp/services/source-code.

Paper in this product is recyclable

To Ekaterina, Alexandra, Kirill, and Maria

Table of Contents

About the Author

Dmitry Vostokov is an internationally recognized expert, speaker, educator, scientist, inventor, and author. He founded the pattern-oriented software diagnostics, forensics, and prognostics discipline (Systematic Software Diagnostics) and Software Diagnostics Institute (DA+TA: DumpAnalysis.org + TraceAnalysis.org). Vostokov has also authored multiple books on software diagnostics, anomaly detection and analysis, software, and memory forensics, root cause analysis and problem-solving, memory dump analysis, debugging, software trace and log analysis, reverse engineering, and malware analysis. He has over thirty years of experience in software architecture, design, development, and maintenance in various industries, including leadership, technical, and people management roles. In his spare time, he presents multiple topics on Debugging.TV and explores software narratology and its further development as narratology of things and diagnostics of things (DoT), software pathology, and quantum software diagnostics. His current interest areas are theoretical software diagnostics and its mathematical and computer science foundations, application of formal logic, artificial intelligence, machine learning, and data mining to diagnostics and anomaly detection, software diagnostics engineering and diagnostics-driven development, diagnostics workflow, and interaction. Recent interest areas also include cloud native computing, security, automation, functional programming, applications of category theory to software development and big data, and artificial intelligence diagnostics.

About the Technical Reviewer

Krishnendu Dasgupta is currently the Head of Machine Learning at Mondosano GmbH, leading data science initiatives focused on clinical trial recommendations and advanced patient health profiling through disease and drug data. Prior to this role, he co-founded DOCONVID AI, a startup that leveraged applied AI and medical imaging to detect lung abnormalities and neurological disorders.

With a strong background in computer science engineering, Krishnendu has more than a decade of experience in developing solutions and platforms using applied machine learning. His professional trajectory includes key positions at prestigious organizations such as NTT DATA, PwC, and Thoucentric.

Krishnendu's primary research interests include applied AI for graph machine learning, medical imaging, and decentralized privacy-preserving machine learning in healthcare. He also had the opportunity to participate in the esteemed Entrepreneurship and Innovation Bootcamp at the Massachusetts Institute of Technology, cohort of 2018 batch.

Beyond his professional endeavors, Krishnendu actively dedicates his time to research, collaborating with various research NGOs and universities worldwide. His focus is on applied AI and ML.

Introduction

Python is the dominant language used in AI and machine learning with data and pipelines in cloud environments. Besides debugging Python code in popular IDEs, notebooks, and command-line debuggers, this book also includes coverage of native OS interfacing (Windows and Linux) necessary to understand, diagnose, and debug complex software issues.

The book begins with an introduction to pattern-oriented software diagnostics and debugging processes that, before doing Python debugging, diagnose problems in various software artifacts such as memory dumps, traces, and logs. Next, it teaches various debugging patterns using Python case studies that model abnormal software behavior. Further, it covers Python debugging specifics in cloud native and machine learning environments. It concludes with how recent advances in AI/ML can help in Python debugging. The book also goes deep for case studies when there are environmental problems, crashes, hangs, resource spikes, leaks, and performance degradation. It includes tracing and logging besides memory dumps and their analysis using native WinDbg and GDB debuggers.

This book is for those who wish to understand how Python debugging is and can be used to develop robust and reliable AI, machine learning, and cloud computing software. It uses a novel pattern-oriented approach to diagnosing and debugging abnormal software structure and behavior. Software developers, AI/ML engineers, researchers, data engineers, MLOps, DevOps, and anyone who uses Python will benefit from this book.

Source Code: All source code used in this book can be downloaded from `github.com/Apress/Python-Debugging-for-AI-Machine-Learning-and-Cloud-Computing`.

CHAPTER 1

Fundamental Vocabulary

Debugging complex software issues in machine learning and cloud computing environments requires not only the knowledge of the Python language and its interpreter (or compiler), plus standard and external libraries, but also necessary and relevant execution environment and operating system internals. In this chapter, you will review some necessary fundamentals from software diagnostics and debugging languages to have the same base level of understanding for the following chapters. In this book, I assume that you are familiar with the Python language and its runtime environment.

Process

A Python script is interpreted by compiling it into bytecode and then executing it, or it can even be precompiled into an application program. In both cases, this interpreter file or the compiled application is an executable program (in Windows, it may have a `.exe` extension) that references some operating system libraries (`.dll` in Windows and .so in Linux). This application can be loaded into computer memory several times; each time, a separate process is created with its own resources and unique process ID (PID, also TGID), as shown in Figure 1-1. The process may also have a parent process that created it, with a parent process ID (PPID).

D. Vostokov, *Python Debugging for AI, Machine Learning, and Cloud Computing*,
https://doi.org/10.1007/978-1-4842-9745-2_1

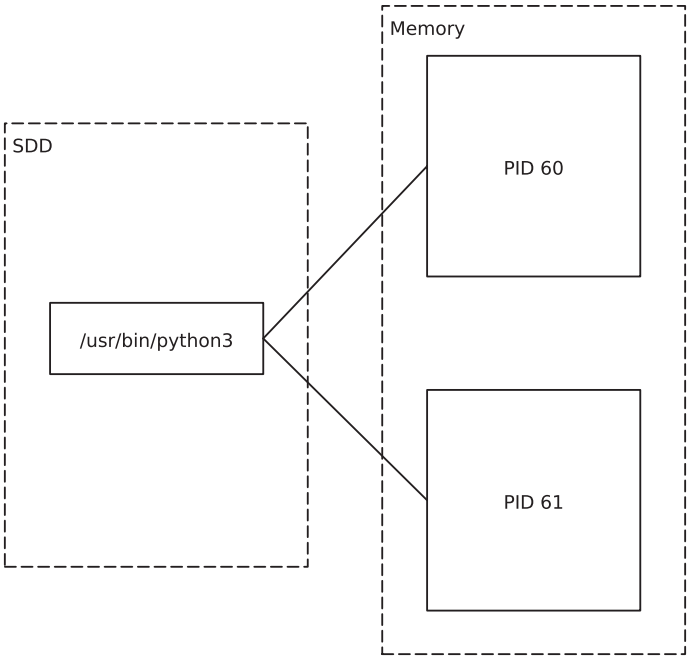

Figure 1-1. *Two python3 processes with two different PIDs*

To illustrate, I executed the code in Listing 1-1 on both Windows and Linux twice.

Listing 1-1. A Simple Script to Model Running Python Code

```
import time

def main():
    foo()

def foo():
    bar()

def bar():
    while True:
        time.sleep(1)

if __name__ == "__main__":
    main()
```

Figure 1-2 shows two processes on Windows.

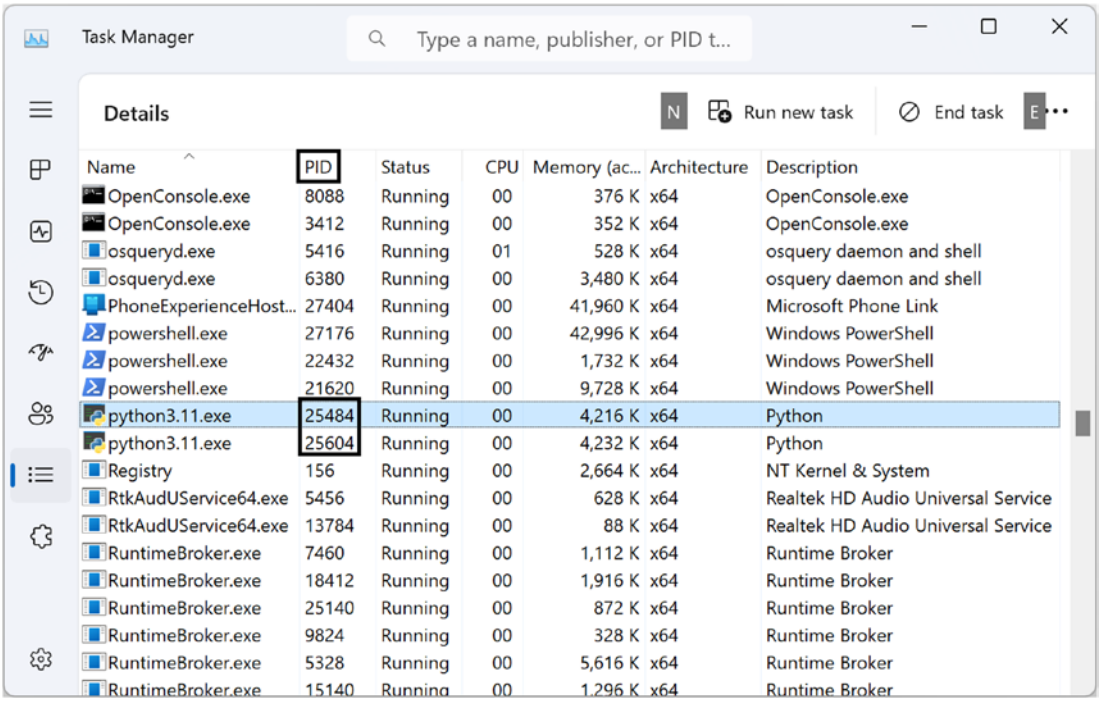

Figure 1-2. *Two running python3.11.exe processes on Windows*

On Linux, you can also see two processes when you execute the same script in two separate terminals:

```
~/Chapter1$ which python3
/usr/bin/python3

~/Chapter1$ ps -a
  PID TTY          TIME CMD
   17 pts/0    00:00:00 mc
   60 pts/2    00:00:00 python3
   61 pts/1    00:00:00 python3
   80 pts/3    00:00:00 ps
```

Note The operating system controls hardware and processes/threads. From a high level, it is just a collection of processes with the operating system kernel as a process too.

Thread

From an operating system perspective, a process is just a memory container for a Python interpreter, its code, and data. But the interpreter code needs to be executed, for example, to interpret the Python bytecode. This unit of execution is called a thread. A process may have several such units of execution (several threads, the so-called multithreaded application). Each thread has its own unique thread ID (TID, also LWP or SPID), as shown in Figure 1-3. For example, one thread may process user interface events and others may do complex calculations in response to UI requests, thus making the UI responsive. On Windows, thread IDs are usually different from process IDs, but in Linux, the thread ID of the main thread is the same as the process ID for a single-threaded process.

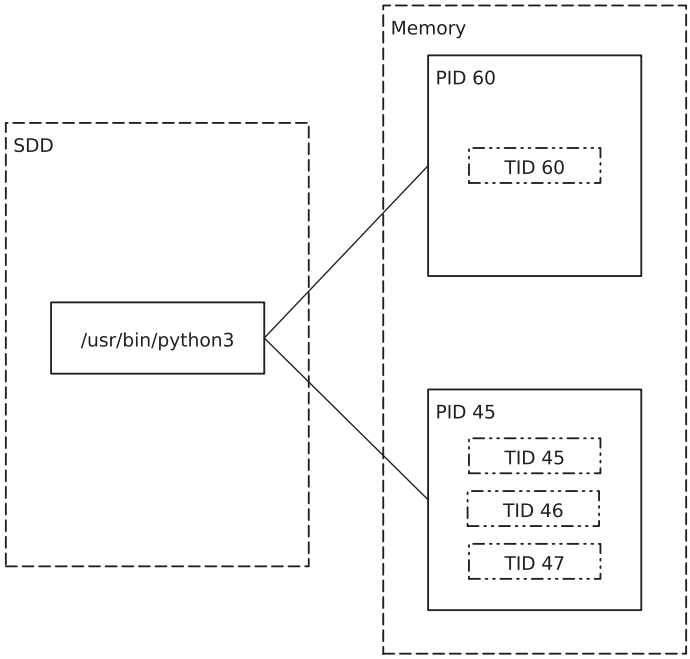

Figure 1-3. *Two python3 processes with different numbers of threads*

To model multithreading, I executed the code in Listing 1-2 on both Windows and Linux.

Listing 1-2. A Simple Script to Model Multiple Threads

```
import time
import threading

def thread_func():
    foo()

def main():
    t1 = threading.Thread(target=thread_func)
    t1.start()
    t2 = threading.Thread(target=thread_func)
    t2.start()
    t1.join()
    t2.join()

def foo():
    bar()

def bar():
    while True:
        time.sleep(1)

if __name__ == "__main__":
    main()
```

Figure 1-4 shows that in Windows, you can see 11 threads at the beginning (this number later changes to 7 and then to 5). You see that the number of threads may be greater than expected.

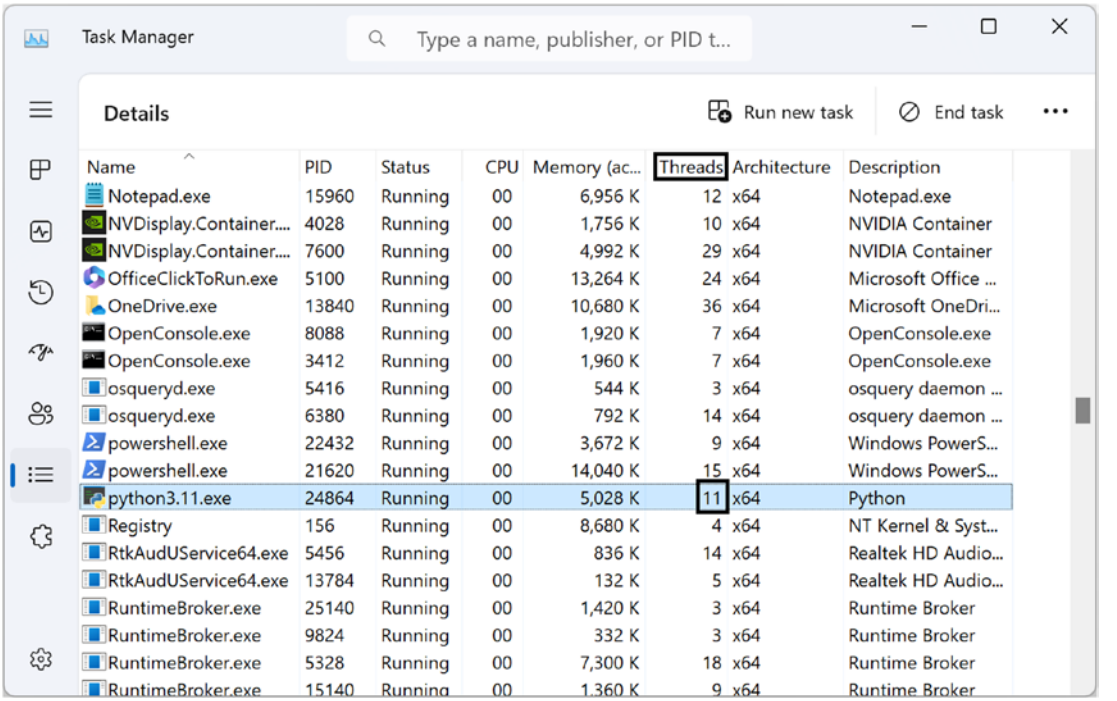

Figure 1-4. *The number of threads in the running python3.11.exe process on Windows*

In Linux, you can see the expected number of threads – 3:

```
~/Chapter1$ ps -aT
  PID  SPID TTY          TIME CMD
   17    17 pts/0    00:00:00 mc
   45    45 pts/2    00:00:00 python3
   45    46 pts/2    00:00:00 python3
   45    47 pts/2    00:00:00 python3
   54    54 pts/1    00:00:00 ps
```

Stack Trace (Backtrace, Traceback)

I should distinguish Python source code tracebacks (which we call *managed stack traces*) and unmanaged (native) ones from the Python compiler and interpreter that compiles to and executes Python byte code. You will see this distinction in some chapters for several case studies and how to get both traces. But, for now, I will just show the difference.