

# 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  
<https://doi.org/10.1007/978-1-4842-9745-2>

ISBN-13 (electronic): 978-1-4842-9745-2

Copyright © 2024 by Dmitry Vostokov

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

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](http://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](mailto:booktranslations@springernature.com); for reprint, paperback, or audio rights, please e-mail [bookpermissions@springernature.com](mailto: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](http://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

<b>About the Author .....</b>	<b>xvii</b>
<b>About the Technical Reviewer .....</b>	<b>xix</b>
<b>Introduction .....</b>	<b>xxi</b>
<b>Chapter 1: Fundamental Vocabulary .....</b>	<b>1</b>
Process .....	1
Thread .....	4
Stack Trace (Backtrace, Traceback) .....	6
Symbol Files.....	12
Module .....	14
Memory Dump .....	16
Crash.....	17
Hang.....	18
Summary.....	20
<b>Chapter 2: Pattern-Oriented Debugging .....</b>	<b>21</b>
The History of the Idea.....	21
Patterns and Analysis Patterns .....	22
Development Process .....	22
Development Patterns.....	23
Debugging Process and Patterns.....	24
Elementary Diagnostics Patterns .....	25
Debugging Analysis Patterns.....	26
Debugging Architecture Patterns .....	26
Debugging Design Patterns.....	26

TABLE OF CONTENTS

- Debugging Implementation Patterns ..... 27
- Debugging Usage Patterns ..... 27
- Debugging Presentation Patterns ..... 27
- Summary ..... 28
- Chapter 3: Elementary Diagnostics Patterns ..... 29**
- Functional Patterns ..... 30
- Use-Case Deviation ..... 30
- Non-Functional Patterns ..... 30
- Crash ..... 30
- Hang ..... 31
- Counter Value ..... 33
- Error Message ..... 34
- Summary ..... 34
- Chapter 4: Debugging Analysis Patterns ..... 35**
- Paratext ..... 36
- State Dump ..... 37
- Counter Value ..... 37
- Stack Trace Patterns ..... 37
- Stack Trace ..... 38
- Runtime Thread ..... 38
- Managed Stack Trace ..... 38
- Source Stack Trace ..... 41
- Stack Trace Collection ..... 41
- Stack Trace Set ..... 41
- Exception Patterns ..... 41
- Managed Code Exception ..... 42
- Nested Exception ..... 42
- Exception Stack Trace ..... 43
- Software Exception ..... 43

Module Patterns.....	43
Module Collection.....	44
Not My Version.....	47
Exception Module.....	47
Origin Module.....	47
Thread Patterns.....	47
Spiking Thread.....	48
Active Thread.....	48
Blocked Thread.....	48
Blocking Module.....	48
Synchronization Patterns.....	48
Wait Chain.....	49
Deadlock.....	49
Livelock.....	49
Memory Consumption Patterns.....	49
Memory Leak.....	49
Handle Leak.....	49
Case Study.....	50
Summary.....	64
<b>Chapter 5: Debugging Implementation Patterns.....</b>	<b>65</b>
Overview of Patterns.....	66
Break-Ins.....	66
Code Breakpoint.....	70
Code Trace.....	71
Scope.....	73
Variable Value.....	75
Type Structure.....	76
Breakpoint Action.....	78
Usage Trace.....	81

TABLE OF CONTENTS

- Case Study ..... 81
  - Elementary Diagnostics Patterns ..... 81
  - Debugging Analysis Patterns..... 81
  - Debugging Implementation Patterns..... 82
- Summary..... 89
- Chapter 6: IDE Debugging in the Cloud ..... 91**
  - Visual Studio Code ..... 91
    - WSL Setup ..... 91
    - Cloud SSH Setup..... 92
  - Case Study..... 96
- Summary..... 109
- Chapter 7: Debugging Presentation Patterns ..... 111**
  - Python Debugging Engines ..... 111
  - Case Study ..... 112
  - Suggested Presentation Patterns ..... 125
- Summary..... 125
- Chapter 8: Debugging Architecture Patterns ..... 127**
  - The Where? Category ..... 128
    - In Papyro..... 129
    - In Vivo ..... 129
    - In Vitro ..... 129
    - In Silico..... 129
    - In Situ ..... 130
    - Ex Situ ..... 130
  - The When? Category ..... 130
    - Live..... 130
    - JIT..... 131
    - Postmortem ..... 131
  - The What? Category ..... 131
    - Code ..... 131

Data .....	132
Interaction .....	132
The How? Category .....	132
Software Narrative .....	132
Software State .....	132
Summary .....	133
<b>Chapter 9: Debugging Design Patterns .....</b>	<b>135</b>
CI Build Case Study .....	137
Elementary Diagnostics .....	137
Analysis .....	138
Architecture .....	138
Design .....	138
Implementation .....	138
Data Processing Case Study .....	138
Elementary Diagnostics .....	139
Analysis .....	139
Architecture .....	146
Design .....	147
Implementation .....	147
Summary .....	147
<b>Chapter 10: Debugging Usage Patterns .....</b>	<b>149</b>
Exact Sequence .....	150
Scripting .....	150
Debugger Extension .....	151
Abstract Command .....	152
Space Translation .....	152
Lifting .....	152
Gestures .....	153
Summary .....	154

TABLE OF CONTENTS

- Chapter 11: Case Study: Resource Leaks ..... 155**
  - Elementary Diagnostics ..... 155
  - Debugging Analysis..... 156
  - Debugging Architecture ..... 160
  - Debugging Implementation..... 161
  - Summary..... 166
  
- Chapter 12: Case Study: Deadlock..... 167**
  - Elementary Diagnostics ..... 167
  - Debugging Analysis..... 168
  - Debugging Architecture ..... 171
  - Exceptions and Deadlocks ..... 173
  - Summary..... 174
  
- Chapter 13: Challenges of Python Debugging in Cloud Computing ..... 175**
  - Complex Distributed Systems ..... 175
    - Granularity of Services ..... 176
    - Communication Channels Overhead..... 176
    - Inter-Service Dependencies ..... 177
  - Layers of Abstraction ..... 178
    - Opaque Managed Services..... 178
    - Serverless and Function as a Service ..... 178
    - Container Orchestration Platforms ..... 179
  - Continuous Integration/Continuous Deployment..... 179
    - Pipeline Failures ..... 179
    - Rollbacks and Versioning ..... 180
    - Immutable Infrastructure..... 181
  - Diversity of Cloud Service Models ..... 181
    - Infrastructure as a Service ..... 181
    - Platform as a Service ..... 182
    - Software as a Service ..... 182

Evolving Cloud Platforms .....	182
Adapting to Changes .....	183
Staying Updated .....	183
Environment Parity.....	184
Library and Dependency Disparities.....	184
Configuration Differences.....	184
Underlying Infrastructure Differences .....	185
Service Variabilities .....	185
Limited Visibility.....	185
Transient Resources .....	185
Log Management.....	186
Monitoring and Alerting.....	187
Latency and Network Issues.....	187
Network Instabilities.....	188
Service-to-Service Communication.....	188
Resource Leaks and Performance.....	188
Resource Starvation .....	189
Concurrency Issues.....	189
Race Conditions.....	190
Deadlocks.....	190
Security and Confidentiality.....	190
Debugger Access Control Restrictions .....	190
Sensitive Data Exposure.....	191
Limited Access .....	192
Cost Implications .....	192
Extended Sessions .....	192
Resource Provisioning and Deprovisioning .....	192
Data Transfer and Storage Fees .....	192
State Management.....	193
Stateful Services .....	193
Data Volume .....	193

TABLE OF CONTENTS

- Limited Tooling Compatibility ..... 193
- Versioning Issues ..... 193
  - Deprecations and Changes ..... 193
  - SDK and Library Updates ..... 194
- Real-time Debugging and User Experience ..... 194
- External Service Dependencies ..... 194
  - Dependency Failures ..... 194
  - Rate Limiting and Quotas ..... 194
- Asynchronous Operations ..... 194
  - Flow Tracking ..... 195
  - Error Propagation ..... 195
- Scaling and Load Challenges ..... 195
  - Load-Based Issues ..... 195
  - Resource Contention ..... 195
- Multi-Tenancy Issues ..... 196
  - Resource Contention ..... 196
  - Data Security ..... 196
- Reliability and Redundancy Issues ..... 196
  - Service Failures ..... 196
  - Data Durability ..... 197
- Summary ..... 197
- Chapter 14: Challenges of Python Debugging in AI and Machine Learning ..... 199**
  - The Nature of Defects in AI/ML ..... 199
    - Complexity and Abstraction Layers ..... 200
    - Non-Determinism and Reproducibility ..... 200
    - Large Datasets ..... 200
    - High-Dimensional Data ..... 200
    - Long Training Times ..... 201
    - Real-Time Operation ..... 201
    - Model Interpretability ..... 201

Hardware Challenges .....	201
Version Compatibility and Dependency Hell .....	201
Data Defects .....	202
Inconsistent and Noisy Data .....	202
Data Leakage.....	202
Imbalanced Data.....	202
Data Quality .....	202
Feature Engineering Flaws.....	202
Algorithmic and Model-Specific Defects.....	203
Gradients, Backpropagation, and Automatic Differentiation.....	203
Hyperparameter Tuning .....	203
Overfitting and Underfitting .....	203
Algorithm Choice .....	204
Deep Learning Defects.....	204
Activation and Loss Choices.....	204
Learning Rate .....	204
Implementation Defects.....	204
Tensor Shapes .....	204
Hardware Limitations and Memory .....	204
Custom Code .....	205
Performance Bottlenecks .....	205
Testing and Validation .....	205
Unit Testing.....	205
Model Validation .....	205
Cross-Validation .....	205
Metrics Monitoring .....	206
Visualization for Debugging .....	206
TensorBoard .....	206
Matplotlib and Seaborn .....	206
Model Interpretability .....	206

TABLE OF CONTENTS

- Logging and Monitoring ..... 206
  - Checkpoints..... 206
  - Logging..... 207
  - Alerts ..... 207
  - Error Tracking Platforms..... 207
- Collaborative Debugging ..... 207
  - Forums and Communities ..... 207
  - Peer Review..... 207
- Documentation, Continuous Learning, and Updates ..... 208
  - Maintaining Documentation ..... 208
  - Library Updates ..... 208
  - Continuous Learning..... 208
- Case Study ..... 208
- Summary..... 212
- Chapter 15: What AI and Machine Learning Can Do for Python Debugging ..... 213**
  - Automated Error Detection..... 213
  - Intelligent Code Fix Suggestions..... 213
  - Interaction Through Natural Language Queries ..... 214
  - Visual Debugging Insights..... 214
  - Diagnostics and Anomaly Detection..... 214
  - Augmenting Code Reviews ..... 215
  - Historical Information Analysis and Prognostics..... 215
  - Adaptive Learning and Personalized Debugging Experience..... 216
  - Test Suite Integration and Optimization ..... 216
  - Enhanced Documentation and Resource Suggestions ..... 216
  - Problem Modeling..... 217
  - Generative Debugging Strategy ..... 217
  - Help with In Papyro Debugging..... 217
  - Summary..... 218

<b>Chapter 16: The List of Debugging Patterns .....</b>	<b>219</b>
Elementary Diagnostics Patterns.....	219
Debugging Analysis Patterns .....	219
Debugging Architecture Patterns .....	221
Debugging Design Patterns .....	222
Debugging Implementation Patterns .....	222
Debugging Usage Patterns .....	222
Debugging Presentation Patterns .....	223
<b>Index.....</b>	<b>225</b>

# 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](http://DumpAnalysis.org) + [TraceAnalysis.org](http://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](http://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](https://github.com/Apress/Python-Debugging-for-AI-Machine-Learning-and-Cloud-Computing).

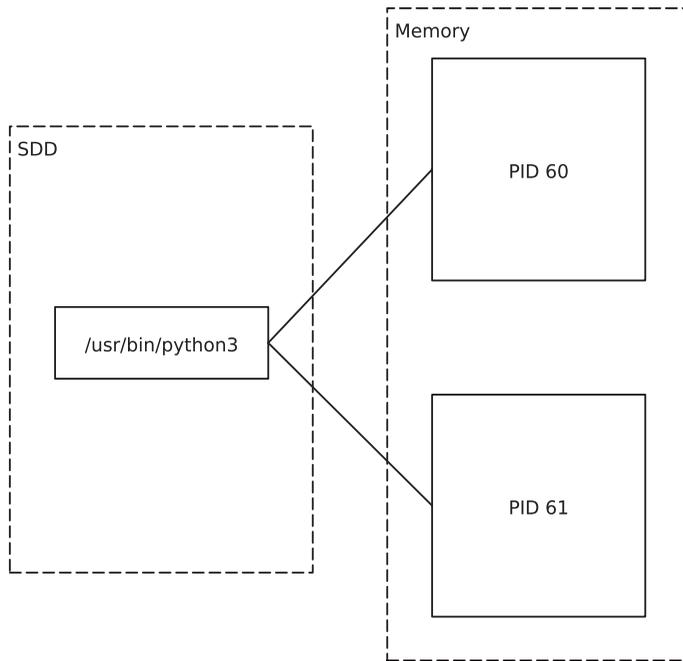
# CHAPTER 1

# 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).



**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

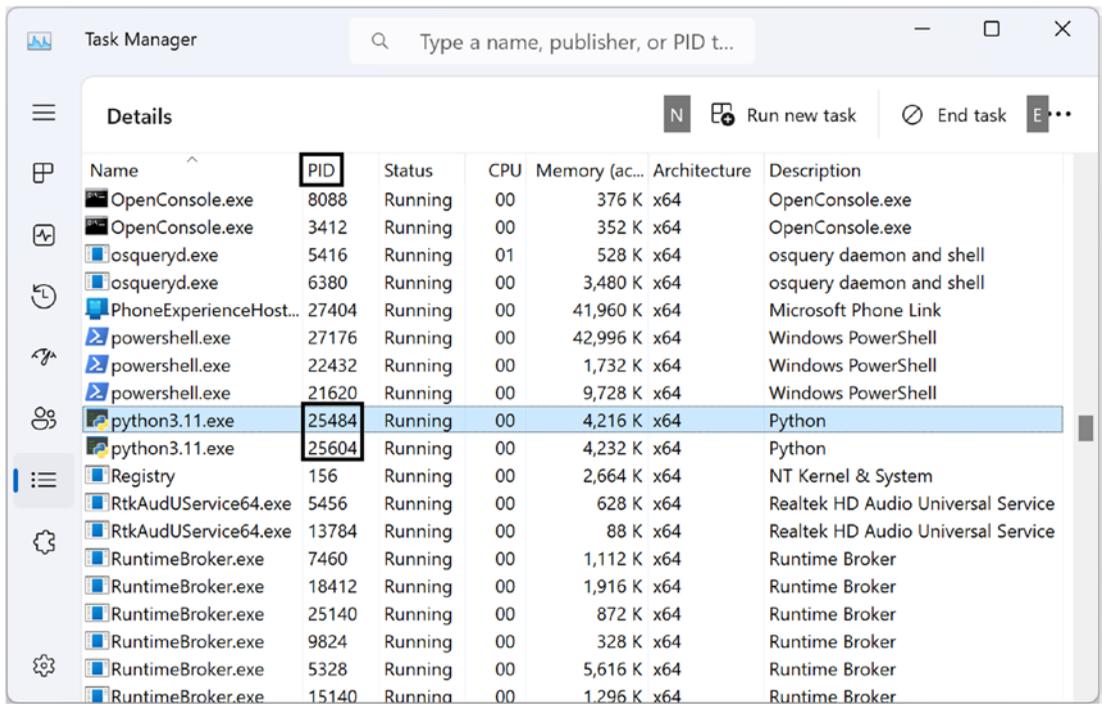
def main():
    foo()

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
```

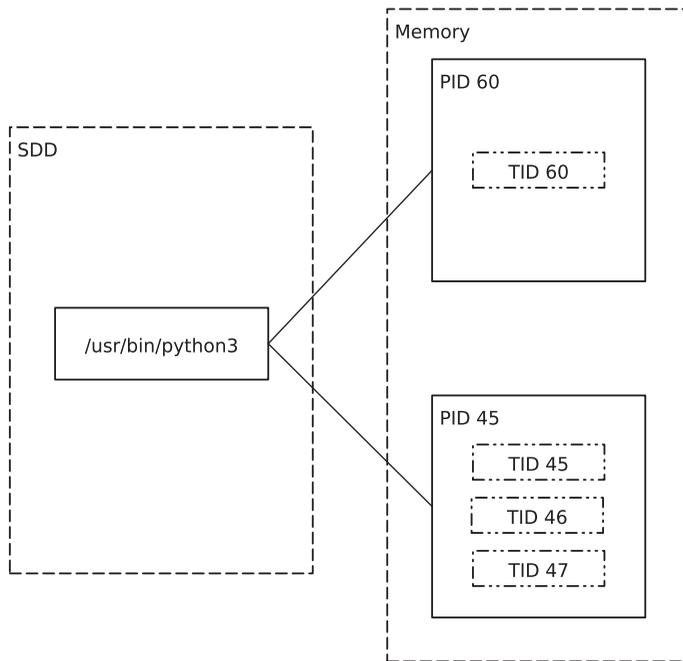
---

**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.

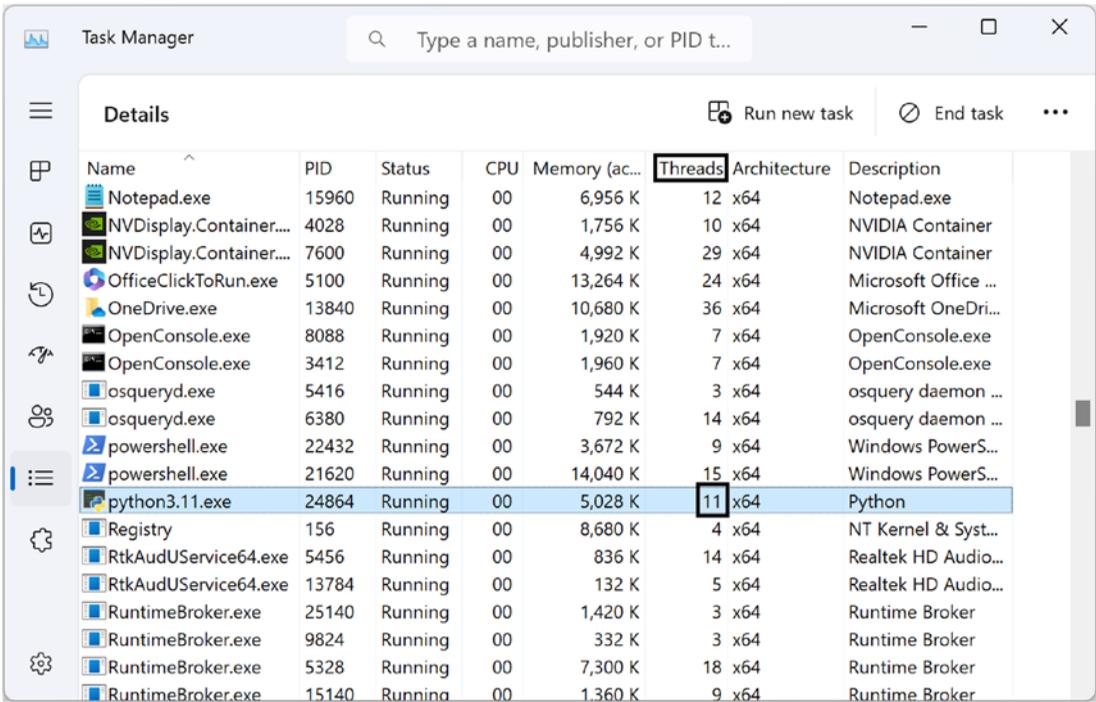


Figure 1-4. The number of threads in the running python3.11.exe process on Windows

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.