Acceptance criteria for the lib/daemon Subplot library

The Subplot project

2021-10-21 11:32

Contents

1 Introduction **1**

2 Daemon is started and terminated **1**

3 Daemon takes a while to open its port **2**

4 Daemon never opens the intended port **2**

5 Daemon stdout and stderr are retrievable **3**

6 Can specify additional environment variables for daemon **3**

1 Introduction

The Subplot[1] library `daemon` for Python provides scenario steps and their implementations for running a background process and terminating at the end of the scenario.

This document explains the acceptance criteria for the library and how they're verified. It uses the steps and functions from the `lib/daemon` library. The scenarios all have the same structure: run a command, then examine the exit code, verify the process is running.

2 Daemon is started and terminated

This scenario starts a background process, verifies it's started, and verifies it's terminated after the scenario ends.

[1] `https://subplot.liw.fi/`

given there is no "**sleep 12765**" process
when I start "**sleep 12765**" as a background process as **sleepyhead**
then a process "**sleep 12765**" is running
when I stop background process **sleepyhead**
then there is no "**sleep 12765**" process

3 Daemon takes a while to open its port

This scenario verifies that if the background process doesn't immediately start listening on its port, the daemon library handles that correctly. We do this with a helper script that waits 2 seconds before opening the port. The lib/daemon code will wait for the script by repeatedly trying to connect. Once successful, it immediately closes the port, which causes the script to terminate.

given a daemon helper shell script **slow-start-daemon.py**
given there is no "**slow-start-daemon.py**" process
when I try to start "**./slow-start-daemon.py**" as **slow-daemon**, on port **8888**
then starting the daemon succeeds
when I stop background process **slow-daemon**
then there is no "**slow-start-daemon.py**" process

File: **slow-start-daemon.py**

```
#!/usr/bin/env python3

import socket
import time

time.sleep(2)

s = socket.socket()
s.bind(("127.0.0.1", 8888))
s.listen()

(conn, _) = s.accept()
conn.recv(1)
s.close()

print("OK")
```

4 Daemon never opens the intended port

This scenario verifies that if the background process never starts listening on its port, the daemon library handles that correctly.

given there is no "**sleep 12765**" process

when I try to start "**sleep 12765**" as **sleepyhead**, on port **8888**
then starting daemon fails with "**ConnectionRefusedError**"
then a process "**sleep 12765**" is running
when I stop background process **sleepyhead**
then there is no "**sleep 12765**" process

5 Daemon stdout and stderr are retrievable

Sometimes it's useful for the step functions to be able to retrieve the stdout or stderr of of the daemon, after it's started, or even after it's terminated. This scenario verifies that `lib/daemon` can do that.

given a daemon helper shell script **chatty-daemon.sh**
given there is no "**chatty-daemon**" process
when I start "**./chatty-daemon.sh**" as a background process as **chatty-daemon**
when daemon **chatty-daemon** has produced output
when I stop background process **chatty-daemon**
then there is no "**chatty-daemon**" process
then daemon **chatty-daemon** stdout is "**hi there\n**"
then daemon **chatty-daemon** stderr is "**hola\n**"

We make for the daemon to exit, to work around a race condition: if the test program retrieves the daemon's output too fast, it may not have had time to produce it yet.

File: **chatty-daemon.sh**

```
#!/usr/bin/env bash

set -euo pipefail

trap 'exit 0' TERM

echo hola 1>&2
echo hi there
```

6 Can specify additional environment variables for daemon

Some daemons are configured through their environment rather than configuration files. This scenario verifies that a step can set arbitrary variables in the daemon's environment.

when I start "**/usr/bin/env**" as a background process as **env**, with environment **{"custom_variable": "has a Value"}**

when daemon **env** has produced output
when I stop background process **env**
then daemon **env** stdout contains "**custom_variable=has a Value**"

given a daemon helper shell script **env-with-port.py**
when I try to start "**./env-with-port.py 8765**" as **env-with-port**, on port **8765**, with environment **{"custom_variable": "1337"}**
when I stop background process **env-with-port**
then daemon **env-with-port** stdout contains "**custom_variable=1337**"

given a daemon helper shell script **env-with-port.py**
when I start "**./env-with-port.py 8766**" as a background process as **another-env-with-port**, on port **8766**, with environment **{"subplot2": "000"}**
when daemon **another-env-with-port** has produced output
when I stop background process **another-env-with-port**
then daemon **another-env-with-port** stdout contains "**subplot2=000**"

It's important that these new environment variables are not inherited by the steps that follow. To verify that, we run one more scenario which *doesn't* set any variables, but checks that none of the variables we mentioned above are present.

when I start "**/usr/bin/env**" as a background process as **env2**
when daemon **env2** has produced output
when I stop background process **env2**
then daemon **env2** stdout doesn't contain "**custom_variable=has a Value**"
then daemon **env2** stdout doesn't contain "**custom_variable=1337**"
then daemon **env2** stdout doesn't contain "**subplot2=000**"

File: **env-with-port.py**

```
#!/usr/bin/env python3

import os
import socket
import sys
import time

for (key, value) in os.environ.items():
    print(f"{key}={value}")

port = int(sys.argv[1])
print(f"port is {port}")

s = socket.socket()
s.bind(("127.0.0.1", port))
s.listen()

(conn, _) = s.accept()
```

```
19 conn.recv(1)
20 s.close()
```