Monkey patching

kaspersky

О себе

Смольников Илья

Команда контроля качества
Kaspersky IoT Secure Gateway

ilya.Smolnikov@Kaspersky.com

Monkey patching

Monkey patching — черная магия Python.

Monkey patching позволяет расширять или изменять поведение библиотек без изменения их кода напрямую.

Посмотрим как мы используем monkey patching в нашем тестовом фреймворке.

https://www.youtube.com/watch?v=XNYalckxCEY

kaspersky

План

Основы monkey patching

Проблема

Патчинг потоков

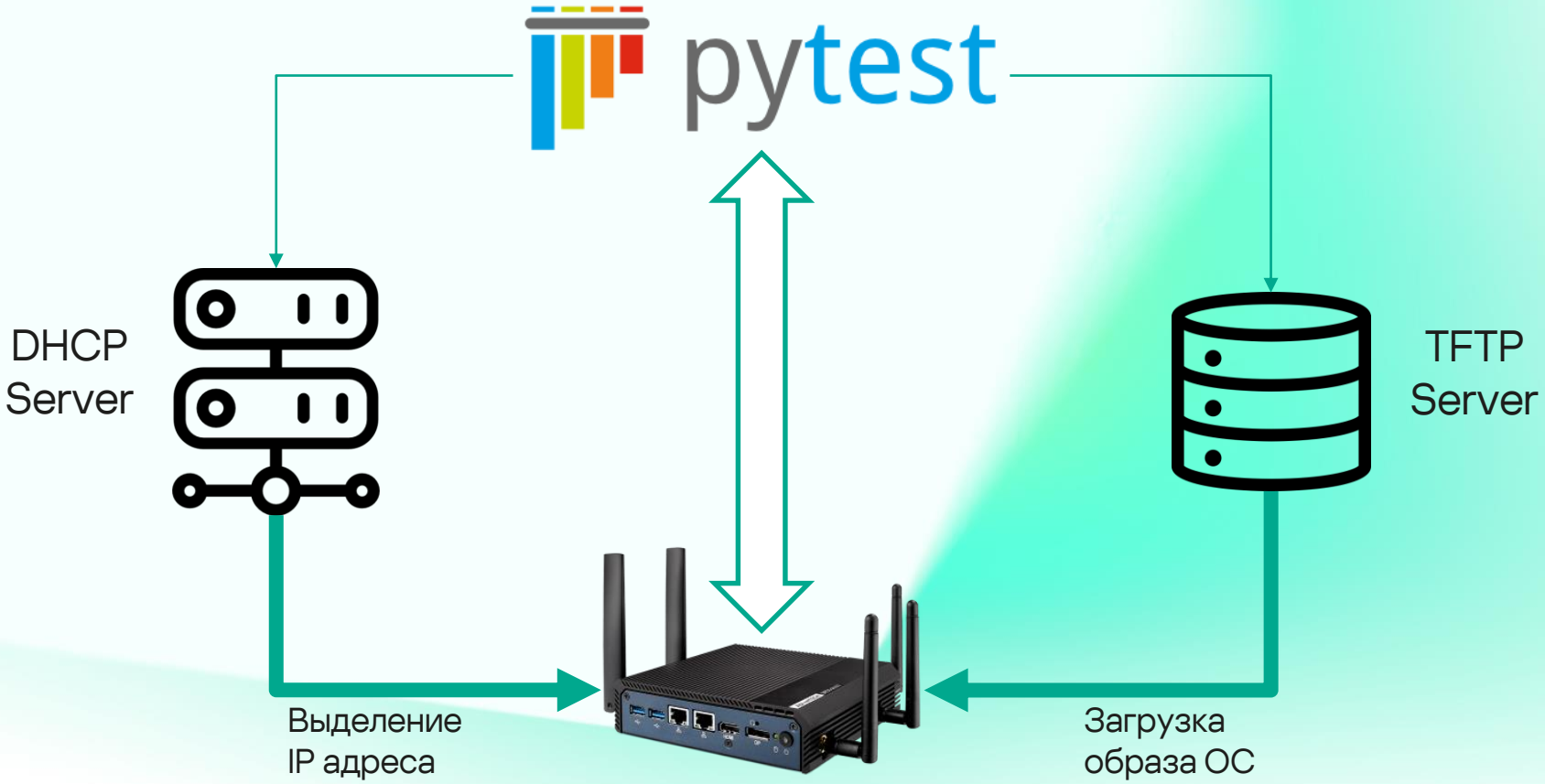
Патчинг процессов

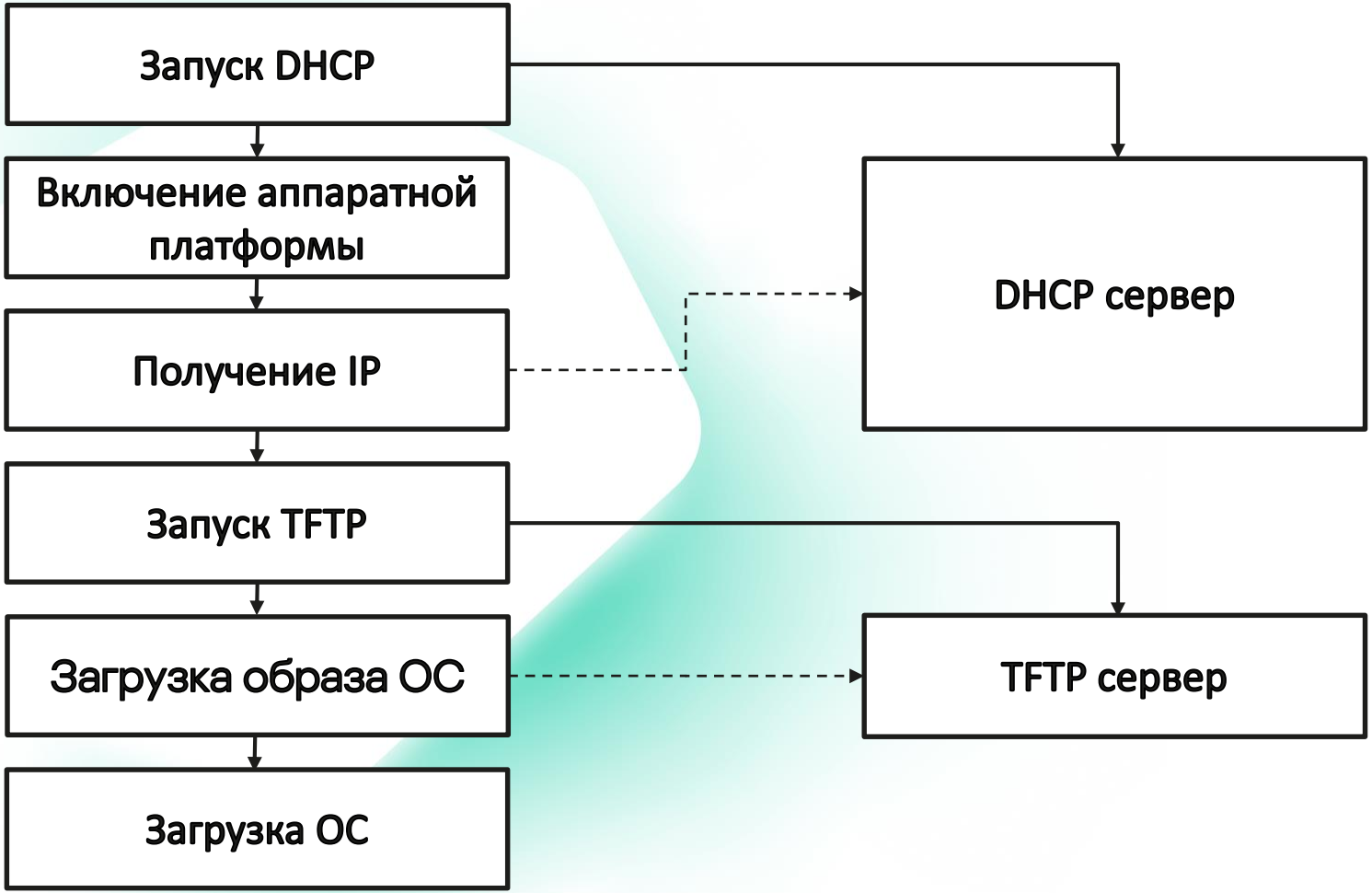
Monkey patching

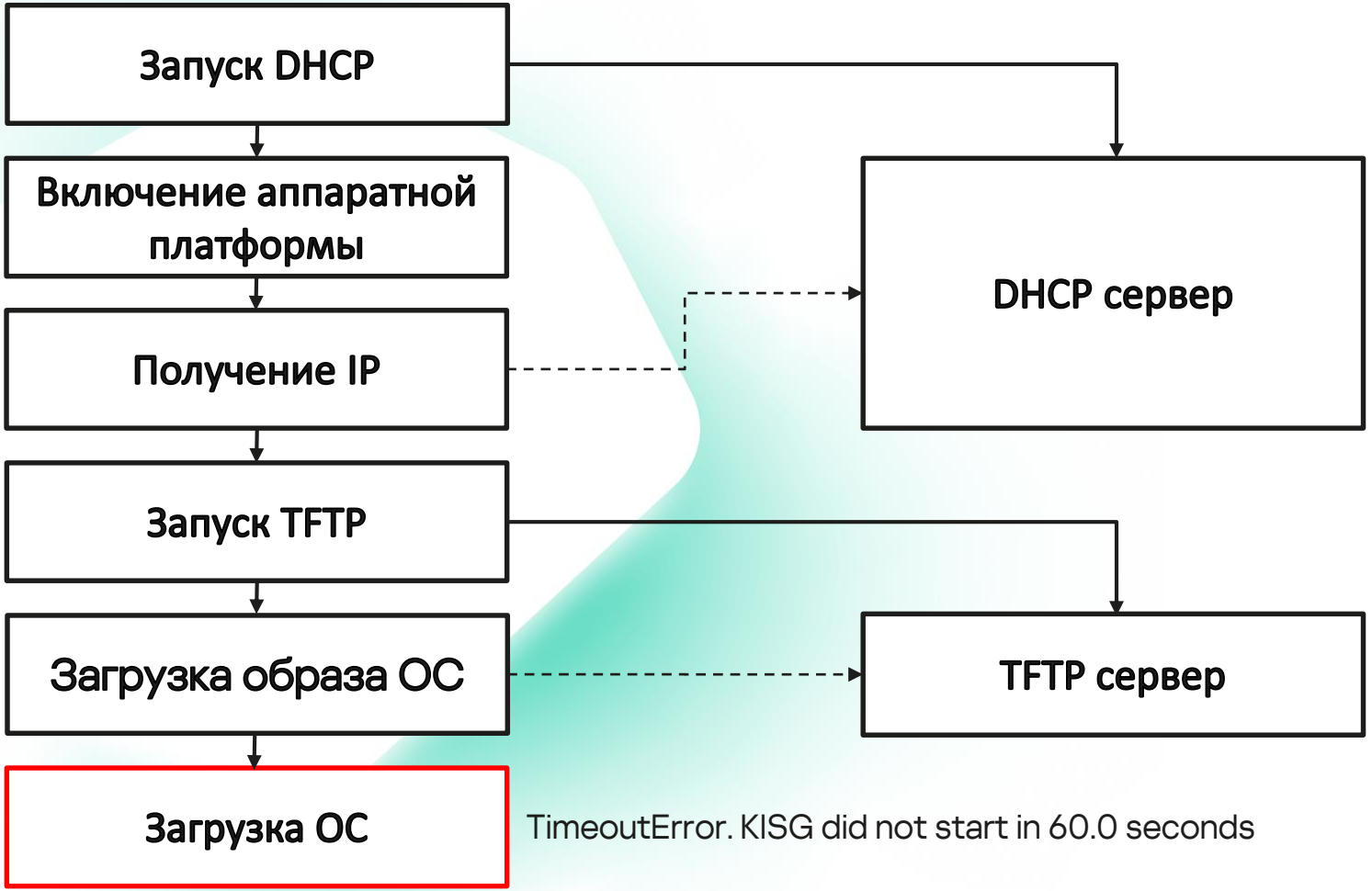

```
1.  import os

2.  def get_current_path():
3.      current_dir = os.getcwd()
4.      return current_dir


5.  def test_func():
6.      def custom_cwd():
7.          return "/data/test“

8.      os.getcwd = custom_cwd
9.      assert get_current_path() == "/data/test“
```

Thread/ process monitor

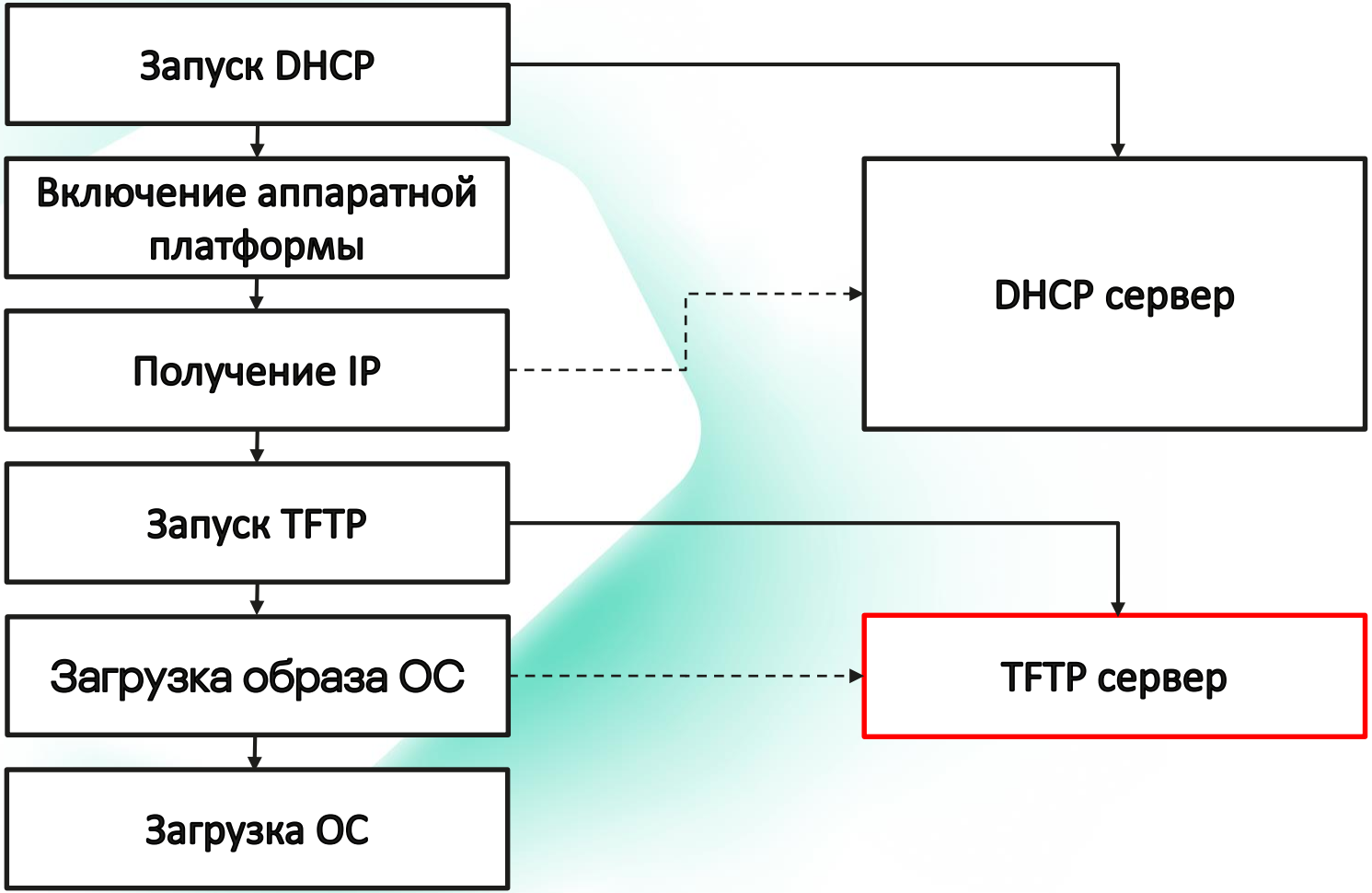
Перехват исключений в дочерних процессах/потоках Python на основе стандартных библиотек


pytest
DHCP
Server
TFTP
Server
Выделение
IP адреса
Загрузка
образа ОС


Запуск DHCP
Включение аппаратной платформы
Получение IP
Запуск TFTP
Загрузка образа ОС
Загрузка ОС
DHCP сервер
TFTP сервер



TimeoutError. KISG did not start in 60.0 seconds


Запуск DHCP
Включение аппаратной платформы
Получение IP
Запуск TFTP
Загрузка образа ОС
Загрузка ОС
DHCP сервер
TFTP сервер

Почему мы выбрали Monkey patching

- Большое количество собственного кода
- Много сторонних библиотек
- Использование в коде стандартных подходов

Thread

Для начала импортируем все,
что нам будет нужно:

```
import os
import signal
import sys
import threading
import traceback
```

Thread

Добавили в класс Thread поле exc_info , в котором для каждого потока будет сохраняться информация об исключении, в случае возникновения:

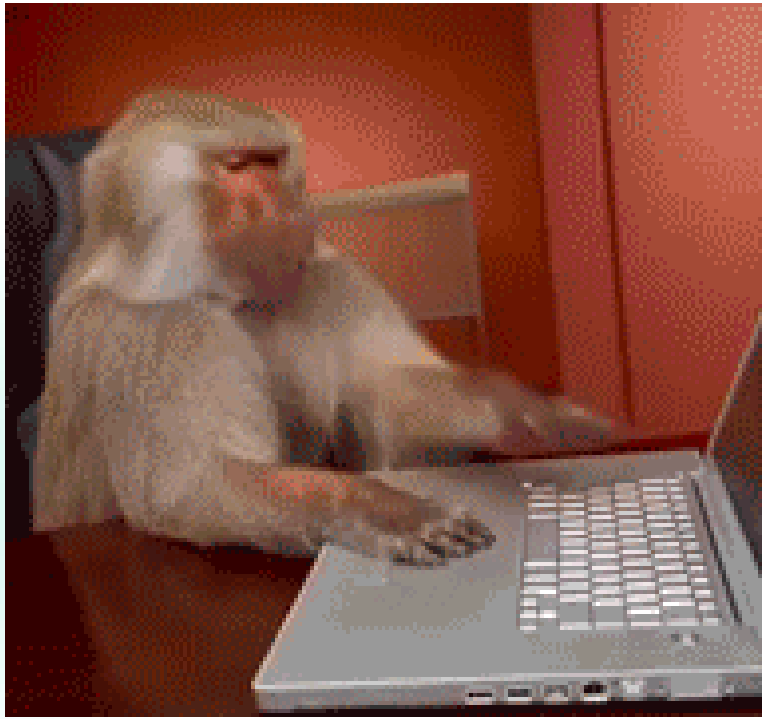
```
setattr(threading.Thread, 'exc_info', None)
```

Информация об ислючении будет сохраняться в виде кортежа:

- имя теста, во время которого случилась неприятность (мы используем pytest и нам важно знать в каком тесте все пошло не по плану)
- тип исключения
- экземпляр исключения
- трейсбэк

Thread

- Патчим __init__
- Патчим run
- Регистрируем обработчик сигналов



Thread


```
1.  init_original = threading.Thread.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
5.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

6.      run_original = self.run
7.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
8.          ...

9.      self.run = run_with_catching_unhandled_exceptions

10. threading.Thread.__init__ = init
```

Thread


```
1.  init_original = threading.Thread.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
5.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

6.      run_original = self.run
7.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
8.          ...

9.      self.run = run_with_catching_unhandled_exceptions

10. threading.Thread.__init__ = init
```

Thread


```
1.  init_original = threading.Thread.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
5.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

6.      run_original = self.run
7.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
8.          ...

9.      self.run = run_with_catching_unhandled_exceptions

10. threading.Thread.__init__ = init
```

Thread


```
1.  init_original = threading.Thread.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
5.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

6.      run_original = self.run
7.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
8.          ...

9.      self.run = run_with_catching_unhandled_exceptions

10. threading.Thread.__init__ = init
```

Thread


```
1.  init_original = threading.Thread.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
5.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

6.      run_original = self.run
7.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
8.          ...

9.      self.run = run_with_catching_unhandled_exceptions

10. threading.Thread.__init__ = init
```

Thread


```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s thread is started by '%s'", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          threading.Thread.exc_info = (self.test_name, *sys.exc_info())
7.          pid = os.getpid()
8.          os.kill(pid, signal.SIGUSR1)
9.      finally:
10.         logging.debug("%s thread is stopped", self.name)
```

Thread


```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s thread is started by '%s'", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          threading.Thread.exc_info = (self.test_name, *sys.exc_info())
7.          pid = os.getpid()
8.          os.kill(pid, signal.SIGUSR1)
9.      finally:
10.         logging.debug("%s thread is stopped", self.name)
```

Thread


```
1.    def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.        try:
3.            logging.debug("%s thread is started by '%s'", self.name, self.trace)
4.            run_original(*args2, **kwargs2)
5.        except:
6.            threading.Thread.exc_info = (self.test_name, *sys.exc_info())
7.            pid = os.getpid()
8.            os.kill(pid, signal.SIGUSR1)
9.        finally:
10.           logging.debug("%s thread is stopped", self.name)
```

Thread


```
1.    def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.        try:
3.            logging.debug("%s thread is started by '%s'", self.name, self.trace)
4.            run_original(*args2, **kwargs2)
5.        except:
6.            threading.Thread.exc_info = (self.test_name, *sys.exc_info())
7.            pid = os.getpid()
8.            os.kill(pid, signal.SIGUSR1)
9.        finally:
10.           logging.debug("%s thread is stopped", self.name)
```

Thread


```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s thread is started by '%s'", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          threading.Thread.exc_info = (self.test_name, *sys.exc_info())
7.          pid = os.getpid()
8.          os.kill(pid, signal.SIGUSR1)
9.      finally:
10.         logging.debug("%s thread is stopped", self.name)
```

Thread


```
1.    def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.        try:
3.            logging.debug("%s thread is started by '%s'", self.name, self.trace)
4.            run_original(*args2, **kwargs2)
5.        except:
6.            threading.Thread.exc_info = (self.test_name, *sys.exc_info())
7.            # Workaround for release thread lock.
8.            try:
9.                lock = self._tstate_lock
10.               if not self.daemon:
11.                   with threading._shutdown_locks_lock:
12.                       threading._shutdown_locks.discard(lock)
13.           except AttributeError as error:
14.               logging.debug("Failed to clean thread lock with error: %s", error)
15.           pid = os.getpid()
16.           os.kill(pid, signal.SIGUSR1)
17.       finally:
18.           logging.debug("%s thread is stopped", self.name)
```

Thread

Обратите внимание на вызов os.kill.
Тут мы отправляем на родительский процесс сигнал (в данном случае используем зарезервированный пользовательский сигнал).

Thread


```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      if threading.Thread.exc_info is not None:
5.          logging.debug("Exception in '%s' class object", threading.Thread.__name__)
6.          test_name, exc_type, exc, exc_traceback = threading.Thread.exc_info
7.          raise exc.with_traceback(exc_traceback)
8.      exit(signum)

9.  signal.signal(signal.SIGUSR1, signal_handler)
```

Thread


```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      if threading.Thread.exc_info is not None:
5.          logging.debug("Exception in '%s' class object", threading.Thread.__name__)
6.          test_name, exc_type, exc, exc_traceback = threading.Thread.exc_info
7.          raise exc.with_traceback(exc_traceback)
8.      exit(signum)

9.  signal.signal(signal.SIGUSR1, signal_handler)
```

Thread


```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      if threading.Thread.exc_info is not None:
5.          logging.debug("Exception in '%s' class object", threading.Thread.__name__)
6.          test_name, exc_type, exc, exc_traceback = threading.Thread.exc_info
7.          raise exc.with_traceback(exc_traceback)
8.      exit(signum)

9.  signal.signal(signal.SIGUSR1, signal_handler)
```

Thread


```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      if threading.Thread.exc_info is not None:
5.          logging.debug("Exception in '%s' class object", threading.Thread.__name__)
6.          test_name, exc_type, exc, exc_traceback = threading.Thread.exc_info
7.          raise exc.with_traceback(exc_traceback)
8.      exit(signum)

9.  signal.signal(signal.SIGUSR1, signal_handler)
```

Thread


```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      if threading.Thread.exc_info is not None:
5.          logging.debug("Exception in '%s' class object", threading.Thread.__name__)
6.          test_name, exc_type, exc, exc_traceback = threading.Thread.exc_info
7.          raise exc.with_traceback(exc_traceback)
8.      exit(signum)

9.  signal.signal(signal.SIGUSR1, signal_handler)
```

Thread


```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      if threading.Thread.exc_info is not None:
5.          logging.debug("Exception in '%s' class object", threading.Thread.__name__)
6.          test_name, exc_type, exc, exc_traceback = threading.Thread.exc_info
7.          raise exc.with_traceback(exc_traceback)
8.      exit(signum)

9.  signal.signal(signal.SIGUSR1, signal_handler)
```

Thread

Как видно мы подменяем оригинальные методы в самом классе Thread. Теперь при создании экземпляра класса и запуске выполнения потока будет выполняться наш код. В этом и заключается суть **monkey patching**.

Process

Для процессов нельзя передавать данные через переменную класса

Для синхронизации данных будем каналы (pipe)

Process

- Патчим __init__
- Патчим run
- Регистрируем обработчик сигналов

Process


```
1.  init_original = multiprocessing.Process.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.out_pipe, self.in_pipe = multiprocessing.Pipe(duplex=False)
5.      self.shutdown_event = multiprocessing.Event()
6.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
7.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

8.      run_original = self.run
9.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
10.         ...

11.     self.run = run_with_catching_unhandled_exceptions

12. multiprocessing.Process.__init__ = init
```

Process


```
1.  init_original = multiprocessing.Process.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.out_pipe, self.in_pipe = multiprocessing.Pipe(duplex=False)
5.      self.shutdown_event = multiprocessing.Event()
6.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
7.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

8.      run_original = self.run
9.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
10.         ...

11.     self.run = run_with_catching_unhandled_exceptions

12. multiprocessing.Process.__init__ = init
```

Process


```
1.  init_original = multiprocessing.Process.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.out_pipe, self.in_pipe = multiprocessing.Pipe(duplex=False)
5.      self.shutdown_event = multiprocessing.Event()
6.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
7.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

8.      run_original = self.run
9.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
10.         ...

11.     self.run = run_with_catching_unhandled_exceptions

12. multiprocessing.Process.__init__ = init
```

Process

```
1.  init_original = multiprocessing.Process.__init__

2.  def init(self, *args, **kwargs):
3.      init_original(self, *args, **kwargs)
4.      self.out_pipe, self.in_pipe = multiprocessing.Pipe(duplex=False)
5.      self.shutdown_event = multiprocessing.Event()
6.      self.test_name = os.getenv("PYTEST_CURRENT_TEST")
7.      self.trace = traceback.format_stack()[-3].split("\n")[0].strip()

8.      run_original = self.run
9.      def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
10.         ...

11.     self.run = run_with_catching_unhandled_exceptions

12. multiprocessing.Process.__init__ = init
```

Process

```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s process is started by '%s' test", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          exc_type, exc, exc_traceback = sys.exc_info()
7.          exc_msg = "".join(exc.args)
8.          exc_traceback = "".join(traceback.format_tb(exc_traceback))
9.          self.in_pipe.send((self.test_name, exc_type, exc_msg, exc_traceback))
10.         pid = os.getppid()
11.         os.kill(pid, signal.SIGUSR2)
12.     finally:
13.         logging.debug("%s process is stopped", self.name)
```

Process


```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s process is started by '%s' test", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          exc_type, exc, exc_traceback = sys.exc_info()
7.          exc_msg = "".join(exc.args)
8.          exc_traceback = "".join(traceback.format_tb(exc_traceback))
9.          self.in_pipe.send((self.test_name, exc_type, exc_msg, exc_traceback))
10.         pid = os.getppid()
11.         os.kill(pid, signal.SIGUSR2)
12.     finally:
13.         logging.debug("%s process is stopped", self.name)
```

Process

```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s process is started by '%s' test", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          exc_type, exc, exc_traceback = sys.exc_info()
7.          exc_msg = "".join(exc.args)
8.          exc_traceback = "".join(traceback.format_tb(exc_traceback))
9.          self.in_pipe.send((self.test_name, exc_type, exc_msg, exc_traceback))
10.         pid = os.getppid()
11.         os.kill(pid, signal.SIGUSR2)
12.     finally:
13.         logging.debug("%s process is stopped", self.name)
```

Process


```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s process is started by '%s' test", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          exc_type, exc, exc_traceback = sys.exc_info()
7.          exc_msg = "".join(exc.args)
8.          exc_traceback = "".join(traceback.format_tb(exc_traceback))
9.          self.in_pipe.send((self.test_name, exc_type, exc_msg, exc_traceback))
10.         pid = os.getppid()
11.         os.kill(pid, signal.SIGUSR2)
12.     finally:
13.         logging.debug("%s process is stopped", self.name)
```

Process

```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s process is started by '%s' test", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          exc_type, exc, exc_traceback = sys.exc_info()
7.          exc_msg = "".join(exc.args)
8.          exc_traceback = "".join(traceback.format_tb(exc_traceback))
9.          self.in_pipe.send((self.test_name, exc_type, exc_msg, exc_traceback))
10.         pid = os.getppid()
11.         os.kill(pid, signal.SIGUSR2)
12.     finally:
13.         logging.debug("%s process is stopped", self.name)
```

Process

Process


```
1.  def run_with_catching_unhandled_exceptions(*args2, **kwargs2):
2.      try:
3.          logging.debug("%s process is started by '%s' test", self.name, self.trace)
4.          run_original(*args2, **kwargs2)
5.      except:
6.          exc_type, exc, exc_traceback = sys.exc_info()
7.          exc_msg = "".join(exc.args)
8.          exc_traceback = "".join(traceback.format_tb(exc_traceback))
9.          self.in_pipe.send((self.test_name, exc_type, exc_msg, exc_traceback))
10.         pid = os.getppid()
11.         os.kill(pid, signal.SIGUSR2)
12.     finally:
13.         self.shutdown_event.wait(timeout=1)
14.         logging.debug("%s process is stopped", self.name)
```

Process

```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      logging.debug("Exception in '%s' class object", multiprocessing.Process.__name__)
5.      for process in multiprocessing.active_children():
6.          if process.out_pipe.poll():
7.              test_name, exc_type, exc_msg, exc_traceback = process.out_pipe.recv()
8.              exc_msg = f"{exc_msg}\n{exc_traceback}“
9.              process.shutdown_event.set()
10.             raise exc_type(exc_msg)
11.     exit(signum)

12. signal.signal(signal.SIGUSR2, signal_handler)
```

Process

```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      logging.debug("Exception in '%s' class object", multiprocessing.Process.__name__)
5.      for process in multiprocessing.active_children():
6.          if process.out_pipe.poll():
7.              test_name, exc_type, exc_msg, exc_traceback = process.out_pipe.recv()
8.              exc_msg = f"{exc_msg}\n{exc_traceback}“
9.              process.shutdown_event.set()
10.             raise exc_type(exc_msg)
11.     exit(signum)

12. signal.signal(signal.SIGUSR2, signal_handler)
```

Process

```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      logging.debug("Exception in '%s' class object", multiprocessing.Process.__name__)
5.      for process in multiprocessing.active_children():
6.          if process.out_pipe.poll():
7.              test_name, exc_type, exc_msg, exc_traceback = process.out_pipe.recv()
8.              exc_msg = f"{exc_msg}\n{exc_traceback}"
9.              process.shutdown_event.set()
10.             raise exc_type(exc_msg)
11.     exit(signum)

12. signal.signal(signal.SIGUSR2, signal_handler)
```

Process


```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      logging.debug("Exception in '%s' class object", multiprocessing.Process.__name__)
5.      for process in multiprocessing.active_children():
6.          if process.out_pipe.poll():
7.              test_name, exc_type, exc_msg, exc_traceback = process.out_pipe.recv()
8.              exc_msg = f"{exc_msg}\n{exc_traceback}“
9.              process.shutdown_event.set()
10.             raise exc_type(exc_msg)
11.     exit(signum)

12. signal.signal(signal.SIGUSR2, signal_handler)
```

Process

```
1.  def signal_handler(signum, frame):
2.      signal_string = signal.strsignal(signum) if hasattr(signal, 'strsignal') else str(signal)
3.      logging.debug("Catch signal '%s' (%d) at %s", signal_string, signum, frame)
4.      logging.debug("Exception in '%s' class object", multiprocessing.Process.__name__)
5.      for process in multiprocessing.active_children():
6.          if process.out_pipe.poll():
7.              test_name, exc_type, exc_msg, exc_traceback = process.out_pipe.recv()
8.              exc_msg = f"{exc_msg}\n{exc_traceback}“
9.              process.shutdown_event.set()
10.             raise exc_type(exc_msg)
11.     exit(signum)

12. signal.signal(signal.SIGUSR2, signal_handler)
```

Process

В целом все похоже на то, что уже сделано для потоков. Отличие только в том, что для передачи информации об исключении мы используем однонаправленный канал.

И вся информация упаковывается в строки.

Результат

Было

```
========================= short test summary info ==========================
ERROR test_kisg.py - TimeoutError. KISG did not start in 60.0 seconds
========================== 1 error in 80.98s (0:01:20) ==========================
```

Результат

Было

```
========================= short test summary info ==========================
ERROR test_kisg.py - TimeoutError. KISG did not start in 60.0 seconds
============================ 1 error in 80.98s (0:01:20) ============================
```

Стало

```
========================= short test summary info ==========================
ERROR test_kisg.py – FileNotFoundError: [Errno 2] No such file or directory: install.sh'
============================ 1 error in 19.28s (0:00:19) ============================
```

Спасибо!

Смольников Илья

Ilya.Smolnikov@Kaspersky.com

kaspersky