



ХОЧУ ПОМОЧЬ
ПРОЕКТУ

Использования обработчиков сигналов

 mango-office.ru



Виртуальная АТС Расширенная

2 000 ₽

Узнать больше

РЕКЛАМА


Статьи


Модуль

Сигналы

Асинхронность

Программирование



 12+

/ Использования обработчиков сигналов

Механизмы для использования обработчиков сигналов в Python.

онной системы, которая позволяет уведомлять программу о событии и обрабатывать его. События могут быть созданы самой системой или отправлены из одного процесса в другой. Поскольку сигналы обрабатываются программой, возможно, что некоторые операции, особенно ввод/вывод, могут привести к ошибкам в середине.

Как и в других формах программирования, основанного на событиях, сигналы принимаются путем создания функции обратного вызова, называемой обработчиком сигнала, которая вызывается при возникновении сигнала. Аргументами обработчика сигнала являются номер сигнала и кадр стека от точки в программе, которая была прервана сигналом.

Основные правила.

Функция `signal.signal()` позволяет определять пользовательские обработчики, которые будут выполняться при получении сигнала. Установлено небольшое количество обработчиков по умолчанию: `signal.SIGPIPE` игнорируется, поэтому об ошибках записи в каналах и сокетах можно сообщать как обычные исключения Python, а `signal.SIGINT` преобразуется в исключение `KeyboardInterrupt`, если родительский процесс не изменил его.

Обработчик определенного сигнала после его установки остается до тех пор, пока он не будет сброшен явным образом, за исключением обработчика для `signal.SIGCHLD`, который следует за базовой реализацией.

Выполнение обработчиков сигналов Python.

Обработчик сигналов Python не выполняется внутри обработчика сигналов низкого уровня (C). Вместо него, низкоуровневый обработчик сигнала устанавливает флаг, который указывает виртуальной машине выполнить соответствующий обработчик сигнала Python на более позднем этапе, например при следующей инструкции байт-кода.

Это имеет последствия:

- Не имеет смысла отлавливать синхронные ошибки, такие как `signal.SIGFPE` или `signal.SIGSEGV`, которые вызваны недопустимой операцией в C-коде. Python вернется из обработчика сигнала к коду C, который, вероятно, снова вызовет тот же сигнал, что приведет к зависанию Python. Начиная с Python 3.3 можно использовать модуль обработчика ошибок, чтобы сообщать о синхронных ошибках.
- Долгосрочные вычисления, реализованные исключительно на языке C, например сопоставление регулярных выражений для большого объема текста, могут выполняться непрерывно в течение произвольного промежутка времени, независимо от полученных сигналов. Обработчики сигналов Python будут вызваны после завершения расчета.

Сигналы и потоки.

Обработчики сигналов Python всегда выполняются в основном потоке Python, даже если сигнал был получен в другом потоке. Это означает, что сигналы не могут использоваться в качестве средства связи между потоками. Вместо этого вы можете использовать примитивы синхронизации из модуля `threading`.

Кроме того, только основной поток может устанавливать новый обработчик сигнала.

Примечание:

- Программирование с обработчиками сигналов Unix - это нетривиальное занятие. Существует некоторая степень фрагментации между версиями Unix, но есть и некоторые различия, поэтому, если возникают какие-то проблемы, необходимо сразу обратиться к документации по операционной системе.

Вверх

Примеры:

Пример программы, которая использует функцию OS [signal.alarm\(\)](#), чтобы ограничить время ожидания открытия файла. Такое поведение полезно, если файл предназначен для последовательного устройства, которое может не включаться, что приводит к зависанию функции [os.open\(\)](#) на неопределенное время. Решение состоит в том, чтобы установить 5-секундную тревогу перед открытием файла. Если операция занимает слишком много времени, сигнал тревоги будет отправлен, и обработчик вызывает исключение.

```
import signal, os

def handler_alarm(signum, frame):
    print('Signal handler called with signal', signum)
    raise OSError("Couldn't open device!")

# Set the signal handler and a 5-second alarm
signal.signal(signal.SIGALRM, handler_alarm)
signal.alarm(5)

# This open() may hang indefinitely
fd = os.open('/dev/ttyS0', os.O_RDWR)

# Отключить alarm
signal.alarm(0)
```

Передача выходных данных программы на такие инструменты, как [head\(1\)](#), приведет к отправке сигнала [signal.SIGPIPE](#) вашему процессу, когда приемник стандартного выхода закроется раньше. Это приводит к исключению, подобному `BrokenPipeError: [Errno 32] Broken pipe`. Чтобы поймать это исключение, оберните точку входа следующим образом:

```
import os, sys

def main():
    try:
        # Симулировать большой вывод
        # (замените этот цикл на ваш код)
        for x in range(10000):
            print("y")
        # здесь сбросим вывод, чтобы запустить
        # внутри блока try - SIGPIPE.
        sys.stdout.flush()
    except BrokenPipeError:
        # Python очищает стандартные потоки при выходе.
        # Перенаправим оставшийся вывод в 'devnull', чтобы
        # избежать другого 'BrokenPipeError' при завершении работы.
        devnull = os.open(os.devnull, os.O_WRONLY)
        os.dup2(devnull, sys.stdout.fileno())
        # Python выходит с кодом ошибки 1 в EPIPE
        sys.exit(1)

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

Не устанавливайте SIGPIPE в SIG_DFL, чтобы избежать [исключения BrokenPipeError](#). Это может привести к неожиданному завершению работы программы, также при каждом прерывании любого соединения с сокетом, когда программа все еще выполняет запись в него.

Содержание раздела:	
• КРАТКИЙ ОБЗОР МАТЕРИАЛА.	
• Сигналы модуля signal	
• Функция alarm() модуля signal	
• Функция getsignal() модуля signal	
• strsignal() модуля signal	
• Функция valid signals() модуля signal	
<div>Вверх</div>	

- [Функция pause\(\) модуля signal](#)
- [Функция pidfd_send_signal\(\) модуля signal](#)
- [Функция pthread_kill\(\) модуля signal](#)
- [Функция pthread_sigmask\(\) модуля signal](#)
- [Функция setitimer\(\) модуля signal](#)
- [Функция set_wakeup_fd\(\) модуля signal](#)
- [Функция siginterrupt\(\) модуля signal](#)
- [Функция signal\(\) модуля signal](#)
- [Функция sigpending\(\) модуля signal](#)
- [Функция sigwait\(\) модуля signal](#)
- [Функция sigwaitinfo\(\) модуля signal](#)
- [Функция sigtimedwait\(\) модуля signal](#)
- [Исключение ItimerError модуля signal](#)
- [Стандартная обработка сигналов OS](#)
- [Таймеры модуля signal](#)