Python.

Многопоточность. GIL. GUI библиотеки.

Процессы и потоки

- Программа называется процессом.
- Процесс при запуске содержит один поток, который обычно называют главным потоком процесса.
- Поток последовательно выполняет инструкции программного кода.

Информация процесса

- Какую информацию хранит процесс?

Информация процесса

- Об объеме занимаемой оперативной памяти
- О списке открытых файлов
- Программный счетчик ссылающийся на выполняемую инструкцию
- Стек вызовов используемый для хранения локальных переменных функции.

Дочерние процессы

- Дочерние процессы являются независимыми и изолированными друг от друга
- У процесса существует флаг, говорящий о том, является ли процесс демоническим
- Демонический процесс автоматически завершается вместе с процессом создавшим его
- Демонические процессы не могут создавать дочерних

Потоки

- Новые потоки в текущем процессе выполняют собственные последовательности инструкций
- Имеют свой стек вызова функций
- Имеют доступ к данным и системным ресурсам выделяемым процессу.

threading

- Lock
- Семафор
- Механизм событий

Пример 1

Зачем нужны потоки?

Зачем нужны потоки?

- для увеличения эффективности использования многоядерной архитектуры современных процессоров, а значит, и производительности программы

- если нам нужно разделить логику работы программы на параллельные полностью или частично асинхронные секции

Global Interpreter Lock

- В каждый момент времени только один поток может исполняться процессором
- Нет борьбы за отдельные переменные
- Исполняемый поток получает доступ по всему окружению

Время работы

Пример 2

Время работы

- Если поток не нуждается в ресурсе CPU — он освобождает GIL
- => в этот момент его может попытаться получить и он сам, и другой поток, и еще и главный поток

Python 3.2

- Улучшенная реализация GIL
- Каждый поток после потери управления ждет небольшой промежуток времени до того, как сможет опять захватить GIL

Многопроцессные приложения

- Модуль subprocess

- subprocess.call(*args*, *stdin=None*, *stdout=None*, *stderr=None*, *shell=False*)

- ***class*** subprocess.Popen(*args*, *stdin=None*, *stdout=None*, *stderr=None*, …)

- потоки, созданные в запускающей программе не забирают GIL

- Сложно передавать данные между процессами

Пример subprocess

Еще один подход

- Модуль multiprocessing
- Очень похож на threading по функциональности
- Другие инструменты синхронизации – Queue и Pipe

- Есть механизм работы с общей памятью – классы Value и Array, которые можно “шарить” между процессами

- Механизм создания пулов процессов

Минусы multiprocessing

- Некоторая платформозависимость, т.к.

в разных ОС работа с процессами происходит по-разному

GUI Библиотеки

- http://wiki.python.org/moin/GuiProgramming

- Tkinter (поставляется с интерпретатором, Tk)
- PyQT (QT)
- PyGtk (GTK)
- wxPython (wxWidgets)

Небольшие решения

- EasyGUI (Tk)
- PyGame
- Pyglet (OpenGL)

Установка сторонних библиотек

- PyPI -- http://pypi.python.org/pypi

- Пакеты носят название «eggs» и имеют расширение .egg

Easy install

- Модуль Python (easy_install), идущий в комплекте библиотеки setuptools, которая позволяет автоматически загружать, собирать, устанавливать и управлять пакетами языка Python

- Установить пакет setuptools

$ wget pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11-py2.7.egg

$ sudo sh setuptools-0.6c11-py2.7.egg

Используем:

sudo easy_install <имя пакета /путь до .egg на диске>

Pip installs python

- pip - это инструмент для установки и управления пакетами Python

$ curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py

$ python get-pip.py

Использование:

$ pip install simplejson

$ pip uninstall simplejson

VirtualEnv

- Инструмент для создания изолированного окружения Python

- $ python virtualenv.py ENV
- $ source bin/activate