Stackless Python -- Continuations On Stage

Christian Tismer
Mission Impossible Software Team
tismer@tismer.com

Overview

This demo will try to explain how continuations work and how they can be used. Having read the paper on Stackless Python is helpful but not mandatory. Visitors are welcome to try these new non-local jump facilities.

Instead of an implementation of coroutines and generators as C extensions, we will see how they can be expressed in Python, using the continuation module. Since the theory of continuations is not very broadly known, a small introduction is given.

Continuations

"The current continuation at any point in the execution of a program is an abstraction of the *rest of the program"*

This abstract definition will be much easier to understand in an interactive session. Visitors are invited to play with continuations as first class objects and learn about the consequences of that sentence.

Generators

Instead of a direct implementation of coroutines and generators, I decided to use the most general approach: Implement continuations as first class callable objects, and express generators and coroutines in Python. These objects can be tried interactively.

Threads vs. Continuations

Few people might know that there is a generator implementation using threads. It can be found in the source distribution under demo/threads/Generator.py. Its performance will be benchmarked against an equivalent implementation using continuations.

Stackless Python Benchmarks

Some benchmarks comparing Stackless Python against standard Python will be presented. Guess which version achieves more PyStones under Windows? ☺

Parallel Pattern Matching

In order to show Stackless Python's power, we will see 10000 tiny tasks working in parallel on a pattern-matching problem. This cannot be done with threads due to their massive memory overhead.

One implementation will be shown that uses explicit scheduler calls. With some luck, we will also see implicit task switching which is planned for Stackless Python 1.1.

Animated Coroutine Transfer

Together with a simple coroutine example, the current tree of stack frames will be visualized by animated graphics.

Stackless Extension Modules

All existing extension modules will also work with Stackless Python. Writing an extension module that allows the new features to be used needs a couple of design considerations. A framework for Stackless Extensions will be presented, together with a working implementation of a module that defines its own interpreter function. Extensions following this pattern have the same flexibility as Python functions.

User-defined functions need no longer be callbacks, but can now be expressed by coroutines, which are often faster at runtime and easier to design.

The *Mission Impossible Software Team* (MI5) will be founded during the conference.