Threads and Processes

“Well, since you last asked us to stop, this thread has moved from discussing languages suitable for professional programmers via accidental users to computer-phobic users. A few more iterations can make this thread really interesting…”

eff-bot, June 1996

Overview

This chapter describes the thread support modules provided with the standard Python interpreter. Note that thread support is optional, and may not be available in your Python interpreter.

This chapter also covers some modules that allow you to run external processes on Unix and Windows systems.

Threads

When you run a Python program, execution starts at the top of the main module, and proceeds downwards. Loops can be used to repeat portions of the program, and function and method calls transfer control to a different part of the program (but only temporarily).

With threads, your program can do several things at one time. Each thread has its own flow of control. While one thread might be reading data from a file, another thread can keep the screen updated.

To keep two threads from accessing the same internal data structure at the same time, Python uses a *global interpreter lock*. Only one thread can execute Python code at the same time; Python automatically switches to the next thread after a short period of time, or when a thread does something that may take a while (like waiting for the next byte to arrive over a network socket, or reading data from a file).

The global lock isn’t enough to avoid problems in your own programs, though. If multiple threads attempt to access the same data object, it may end up in an inconsistent state. Consider a simple cache:

```
def getitem(key):
    item = cache.get(key)
    if item is None:
        # not in cache; create a new one
        item = create_new_item(key)
        cache[key] = item
    return item
```

If two threads call the **getitem** function just after each other with the same missing key, they’re likely to end up calling **create_new_item** twice with the same argument. While this may be okay in many cases, it can cause serious problems in others.

To avoid problems like this, you can use *lock objects* to synchronize threads. A lock object can only be owned by one thread at a time, and can thus be used to make sure that only one thread is executing the code in the **getitem** body at any time.

Processes

On most modern operating systems, each program run in its own *process*. You usually start a new program/process by entering a command to the shell, or by selecting it in a menu. Python also allows you to start new programs from inside a Python program.

Most process-related functions are defined by the **os** module. See the [Working with Processes] section for the full story.

Contents