Platform Specific Modules

Overview

This chapter describes some platform specific modules. I've emphasized modules that are available on entire families of platforms (such as Unix, or the Windows family).

The fcntl module

(Unix only). This module provides an interface to the **ioctl** and **fcntl** functions on Unix. They are used for "out of band" operations on file handles and I/O device handles. This includes things like reading extended attributes, controlling blocking, modifying terminal behavior, etc.

Exactly how to use these functions are highly platform dependent. For more information on what you can do on your platform, check the corresponding Unix manual pages.

This module also provides an interface to Unix' file locking mechanisms. The following example uses the **flock** function to place an *advisory lock* on the file, while it is being updated.

The output shown below was obtained by running three instances of the program in parallel, like this: **python fcntl-example-1.py& python fcntl-example-1.py& python fcntl-example-1.py&** (all on one command line). If you comment out the call to **flock**, the counter will not be updated properly.

Example: Using the fcntl module

```
# File:fcntl-example-1.py

import fcntl, FCNTL
import os, time

FILE = "counter.txt"

if not os.path.exists(FILE):
    # create the counter file if it doesn't exist
    file = open(FILE, "w")
    file.write("0")
    file.close()

for i in range(20):
    # increment the counter
    file = open(FILE, "r+")
    fcntl.flock(file.fileno(), FCNTL.LOCK_EX)
    counter = int(file.readline()) + 1
    file.seek(0)
    file.write(str(counter))
    file.close() # unlocks the file
    print os.getpid(), "=>", counter
    time.sleep(0.1)
```

```
30940 => 1
30942 => 2
30941 => 3
30940 => 4
30941 => 5
30942 => 6
```

The pwd module

(Unix only). This module provides an interface to the Unix password "database" (**/etc/passwd** and friends). This database (usually a plain text file) contains information about the user accounts on the local machine.

Example: Using the pwd module

```
# File:pwd-example-1.py

import pwd
import os

print pwd.getpwuid(os.getgid())
print pwd.getpwnam("root")
```

```
('effbot', 'dsWjk8', 4711, 4711, 'eff-bot', '/home/effbot', '/bin/bosh')
('root', 'hs2giiw', 0, 0, 'root', '/root', '/bin/bash')
```

The **getpwall** function returns a list of database entries for all available users. This can be useful if you want to search for a user.

If you have to look up many names, you can use **getpwall** to preload a dictionary:

Example: Using the pwd module

```
# File:pwd-example-2.py

import pwd
import os

# preload password dictionary
_pwd = {}
for info in pwd.getpwall():
    _pwd[info[0]] = _pwd[info[2]] = info

def userinfo(uid):
    # name or uid integer
    return _pwd[uid]

print userinfo(os.getuid())
print userinfo("root")
```

```
('effbot', 'dsWjk8', 4711, 4711, 'eff-bot', '/home/effbot', '/bin/bosh')
('root', 'hs2giiw', 0, 0, 'root', '/root', '/bin/bash')
```

The grp module

(Unix only). This module provides an interface to the Unix group database (**/etc/group**). The **getgrgid** function returns data for a given group identity, **getgrnam** for a group name.

Example: Using the grp module

```
# File:grp-example-1.py

import grp
import os

print grp.getgrgid(os.getgid())
print grp.getgrnam("wheel")
```

```
('effbot', '', 4711, ['effbot'])
('wheel', '', 10, ['root', 'effbot', 'gorbot', 'timbot'])
```

The **getgrall** function returns a list of database entries for all available groups.

If you're going to do a lot of group queries, you can save some time by using **getgrall** to copy all the (current) groups into a dictionary. The **groupinfo** function in the following example returns the information for either a group identifier (an integer) or a group name (a string):

Example: Using the grp module to cache group information

```
# File:grp-example-2.py

import grp
import os

# preload password dictionary
_grp = {}
for info in grp.getgrall():
    _grp[info[0]] = _grp[info[2]] = info

def groupinfo(gid):
    # name or gid integer
    return _grp[gid]

print groupinfo(os.getgid())
print groupinfo("wheel")
```

```
('effbot', '', 4711, ['effbot'])
('wheel', '', 10, ['root', 'effbot', 'gorbot', 'timbot'])
```

The nis module

(Unix only, Optional). This module provides an interface to the NIS (yellow pages) services. It can be used to fetch values from a central NIS database, if available.

Example: Using the nis module

```
# File:nis-example-1.py

import nis
import string

print nis.cat("ypservers")
print string.split(nis.match("bacon", "hosts.byname"))
```

```
{'bacon.spam.egg': 'bacon.spam.egg'}
['194.18.155.250', 'bacon.spam.egg', 'bacon', 'spam-010']
```

The curses module

(Unix only, Optional). The curses module gives you better control of the text terminal window, in a terminal-independent way.

For more information on the curses module, see Andrew Kuchling's *Curses Programming with Python* (*http://www.amk.ca/python/howto/curses/*). For a console driver for Windows, see my own *Windows Console Driver (http://effbot.org/zone/console-index.htm*).

Example: Using the curses module

```
# File:curses-example-1.py

import curses

text = [
   "a very simple curses demo",
   "",
   "(press any key to exit)"
]

# connect to the screen
screen = curses.initscr()

# setup keyboard
curses.noecho() # no keyboard echo
curses.cbreak() # don't wait for newline

# screen size
rows, columns = screen.getmaxyx()

# draw a border around the screen
screen.border()

# display centered text
y = (rows - len(text)) / 2

for line in text:
   screen.addstr(y, (columns-len(line))/2, line)
   y = y + 1

screen.getch()

curses.endwin()
```

The termios module

(Unix only, Optional). This module provides an interface to the Unix terminal control facilities. It can be used to control most aspects of the terminal communication ports.

In the following example, this module is used to temporarily disable keyboard echo (which is controlled by the **ECHO** flag in the third flag field):

Example: Using the termios module

```
# File:termios-example-1.py

import termios, TERMIOS
import sys

fileno = sys.stdin.fileno()

attr = termios.tcgetattr(fileno)
orig = attr[:]

print "attr =>", attr[:4] # flags

# disable echo flag
attr[3] = attr[3] & ~TERMIOS.ECHO

try:
    termios.tcsetattr(fileno, TERMIOS.TCSADRAIN, attr)
    message = raw_input("enter secret message: ")
    print
finally:
    # restore terminal settings
    termios.tcsetattr(fileno, TERMIOS.TCSADRAIN, orig)

print "secret =>", repr(message)
```

```
attr => [1280, 5, 189, 35387]
enter secret message:
secret => 'and now for something completely different'
```

The tty module

(Unix only). This module contains some utility functions for dealing with tty devices. The following example shows how to switch the terminal window over to "raw" mode, and back again.

Example: Using the tty module

```
# File:tty-example-1.py

import tty
import os, sys

fileno = sys.stdin.fileno()

tty.setraw(fileno)
print raw_input("raw input: ")

tty.setcbreak(fileno)
print raw_input("cbreak input: ")

os.system("stty sane") # ...
```

```
raw input: this is raw input
cbreak input: this is cbreak input
```

The resource module

(Unix only, Optional). This module is used to query or modify the system resource limits.

Example: Using the resource module to query current settings

```
# File:resource-example-1.py

import resource

print "usage stats", "=>", resource.getrusage(resource.RUSAGE_SELF)
print "max cpu", "=>", resource.getrlimit(resource.RLIMIT_CPU)
print "max data", "=>", resource.getrlimit(resource.RLIMIT_DATA)
print "max processes", "=>", resource.getrlimit(resource.RLIMIT_NPROC)
print "page size", "=>", resource.getpagesize()
```

```
usage stats => (0.03, 0.02, 0, 0, 0, 0, 75, 168, 0, 0, 0, 0, 0, 0, 0, 0)
max cpu => (2147483647, 2147483647)
max data => (2147483647, 2147483647)
max processes => (256, 256)
page size => 4096
```

Example: Using the resource module to limit resources

```
# File:resource-example-2.py

import resource

# limit the execution time to one second
resource.setrlimit(resource.RLIMIT_CPU, (0, 1))

# pretend we're busy
for i in range(1000):
    for j in range(1000):
        for k in range(1000):
            pass
```

```
CPU time limit exceeded
```

The syslog module

(Unix only, Optional). This module sends messages to the system logger facility (**syslogd**). Exactly what happens to these messages is system-dependent, but they usually end up in a log file named **/var/log/messages**, **/var/adm/syslog**, or some variation thereof. (If you cannot find it, check with your favorite system administrator.)

Example: Using the syslog module

```
# File:syslog-example-1.py

import syslog
import sys

syslog.openlog(sys.argv[0])

syslog.syslog(syslog.LOG_NOTICE, "a log notice")
syslog.syslog(syslog.LOG_NOTICE, "another log notice: %s" % "watch out!")

syslog.closelog()
```

The msvcrt module

(Windows/DOS only). This module gives you access to a number of functions in the Microsoft Visual C/C++ Runtime Library (MSVCRT).

The **getch** function reads a single keypress from the console:

Example: Using the msvcrt module to get key presses

```
# File:msvcrt-example-1.py

import msvcrt

print "press 'escape' to quit..."

while 1:
    char = msvcrt.getch()
    if char == chr(27):
        break
    print char,
    if char == chr(13):
        print
```

```
press 'escape' to quit...
h e l l o
```

The **kbhit** function returns true if a key has been pressed (which means that **getch** won't block).

Example: Using the msvcrt module to poll the keyboard

```
# File:msvcrt-example-2.py

import msvcrt
import time

print "press SPACE to enter the serial number"

while not msvcrt.kbhit() or msvcrt.getch() != " ":
    # do something else
    print ".",
    time.sleep(0.1)

print

# clear the keyboard buffer
while msvcrt.kbhit():
    msvcrt.getch()

serial = raw_input("enter your serial number: ")

print "serial number is", serial
```

```
press SPACE to enter the serial number
. . . . . .
enter your serial number: 10
serial number is 10
```

The **locking** function can be used to implement cross-process file locking under Windows:

Example: Using the msvcrt module for file locking

```
# File:msvcrt-example-3.py

import msvcrt
import os

LK_UNLCK = 0 # unlock the file region
LK_LOCK = 1 # lock the file region
LK_NBLCK = 2 # non-blocking lock
LK_RLCK = 3 # lock for writing
LK_NBRLCK = 4 # non-blocking lock for writing

FILE = "counter.txt"

if not os.path.exists(FILE):
    file = open(FILE, "w")
    file.write("0")
    file.close()

for i in range(20):
    file = open(FILE, "r+")
    # look from current position (0) to end of file
    msvcrt.locking(file.fileno(), LK_LOCK, os.path.getsize(FILE))
    counter = int(file.readline()) + 1
    file.seek(0)
    file.write(str(counter))
    file.close() # unlocks the file
    print os.getpid(), "=>", counter
    time.sleep(0.1)
```

```
208 => 21
208 => 22
208 => 23
208 => 24
208 => 25
208 => 26
```

The nt module

(Implementation, Windows only). This module is an implementation module used by the **os** module on Windows platforms. There's hardly any reason to use this module directly; use **os** instead.

Example: Using the nt module

```
# File:nt-example-1.py

import nt

# in real life, use os.listdir and os.stat instead!
for file in nt.listdir("."):
    print file, nt.stat(file)[6]
```

```
aifc-example-1.py 314
anydbm-example-1.py 259
array-example-1.py 48
```

The _winreg module

(New in 2.0) This module provides a basic interface to the Windows registry database.

The interface provided by this module closely mirrors the corresponding Win32 API.

Example: Using the _winreg module

```
# File:winreg-example-1.py

import _winreg

explorer = _winreg.OpenKey(
    _winreg.HKEY_CURRENT_USER,
    "Software\\Microsoft\\Windows\\CurrentVersion\\Explorer"
    )

# list values owned by this registry key
try:
    i = 0
    while 1:
        name, value, type = _winreg.EnumValue(explorer, i)
        print repr(name),
        i += 1
except WindowsError:
    print

value, type =_winreg.QueryValueEx(explorer, "Logon User Name")

print
print "user is", repr(value)
```

```
'Logon User Name' 'CleanShutdown' 'ShellState' 'Shutdown Setting'
'Reason Setting' 'FaultCount' 'FaultTime' 'IconUnderline' ...

user is u'Effbot'
```

The posix module

(Implementation, Unix/POSIX only). This module is an implementation module used by the **os** module on Unix and other POSIX systems. While everything in here can be (and should be) accessed via the **os** module, you may wish to explicitly refer to this module in situations where you want to make it clear that you expect POSIX behavior.

Example: Using the posix module

```
# File:posix-example-1.py

import posix

for file in posix.listdir("."):
    print file, posix.stat(file)[6]
```

```
aifc-example-1.py 314
anydbm-example-1.py 259
array-example-1.py 48
```