SUB-PROCESS

Rene Alvarez

Python Subprocess

- Allows you to Spawn a new process
- Replacement of several older modules
  - os.system
  - os.spawn*
  - os.popen*
  - Popen2.*
  - commands.*

Why Subprocess Module

- Unified module
- Cross-process exceptions – exec()
- Handling file decriptor is good
- "pipe" support for connecting several subprocesses
- communicate() method – Producer & Consumer problem – avoid deadlocks

Using It

- subprocess.call(args, *, stdin=None, stdout=None, stderr=None, shell=False)
- subprocess.check_output(args, *, stdin=None, stderr=None, shell=False, universal_newlines=False)
  - Run command with arguments and return its output as a byte string
- subprocess.PIPE
- subprocess.STDOUT
- subprocess.CalledProcessError
- returncode – Exit status of the child process

...Using It

- Popen Constructor
  - Execute a child program in a new process
- Exceptions
  - Can handle easily by Parent
- Security
  - Never call a system shell implicitly

Popen Objects

- Popen.poll()
  - Check if child process has terminated
- Popen.wait()
  - Wait for child process to terminate
- Popen.communicate(input=None)
  - It interact with process
- Popen.send_signal(signal)
- Popen.terminate()
- Popen.kill()

Replacing Older Functions

- Examples
- 1. Replacing /bin/sh

```
>>> output='ls -ltr'
>>> import subprocess
>>> output = subprocess.check_output(["ls", "-ltr"])
>>> print output
```

- 2. Replacing Shell pipeline

```
>>> output='dmesg | grep hda'
>>> p1 = subprocess.Popen(['dmesg'], stdout=subprocess.PIPE)
>>> p2 = subprocess.Popen(['grep', 'hda'], stdin=p1.stdout, stdout=subprocess.PIPE)
>>> print p2.communicate()[0]
```

- 3. Replacing os.system()

```
>>> status = os.system("ls -ltr")
>>> status = subprocess.call("ls -ltr", shell=True)
```