

Programando con Stackless Python

DINAMISMO

La extensión Stackless aporta procesos ligeros a Python, dando paso a un nuevo estilo de programación basado en el acceso a la memoria dinámica (heap). **POR STEPHAN DIEHL**

Stackless Python [1], el módulo creado por Christian Tismer, amplía el popular intérprete Python, añadiendo elementos que facilitan el desarrollo de aplicaciones escalables. Encapsula secciones pequeñas e independientes de los programas en tasklets. Para comunicarse, estas tasklets hacen

uso de canales, de forma nostálgicamente parecida a como se hacía en los lenguajes Erlang [2] y Oz [3]. El nombre Stackless se debe a que las tasklets permiten que se intercambien las funciones encapsuladas de la pila (stack [4]) a la memoria dinámica (heap [5]). En cualquier momento se puede acceder a los

Listado 1: Primeros Pasos

```
01 Python 2.5 Stackless 3.1b3          09 ...
02 060516 (release25-maint:53626,    10 >>> f_task =
03 Feb 3 2007, 15:30:37)              11 stackless.tasklet(f)()
04 [GCC 4.0.3 (Ubuntu                  12 <stackless.tasklet object at
05 4.0.3-1ubuntu5)] on linux2         13 0xb7d50e2c>
06 Type "help", "copyright",          14 >>> stackless.schedule(None)
07 "credits" or "license" for         15 1
08 more information.                   16 >>>
09 >>> import stackless
10 >>> def f():
11 ... print "1"
12 ... stackless.schedule()
13 ... print "2"
```

Instalación

En el momento de escribir esto no había paquetes con binarios de Stackless Python disponibles para ninguna distribución. Tenemos que usar Subversion para descargar la revisión actual. Hay versiones para Python 2.4 y 2.5. Después de la descarga, sólo hay que seguir los pasos habituales para compilar e instalar:

```
./configure --prefix=U
/targetdirectory/
make
make install
ln -s /targetdirectory/bin/
python U
/usr/local/bin/stackless
```

El enlace simbólico evita los posibles conflictos con nuestra actual instalación de Python. Para más detalles, consultar la documentación de Stackless en la página de inicio del proyecto.

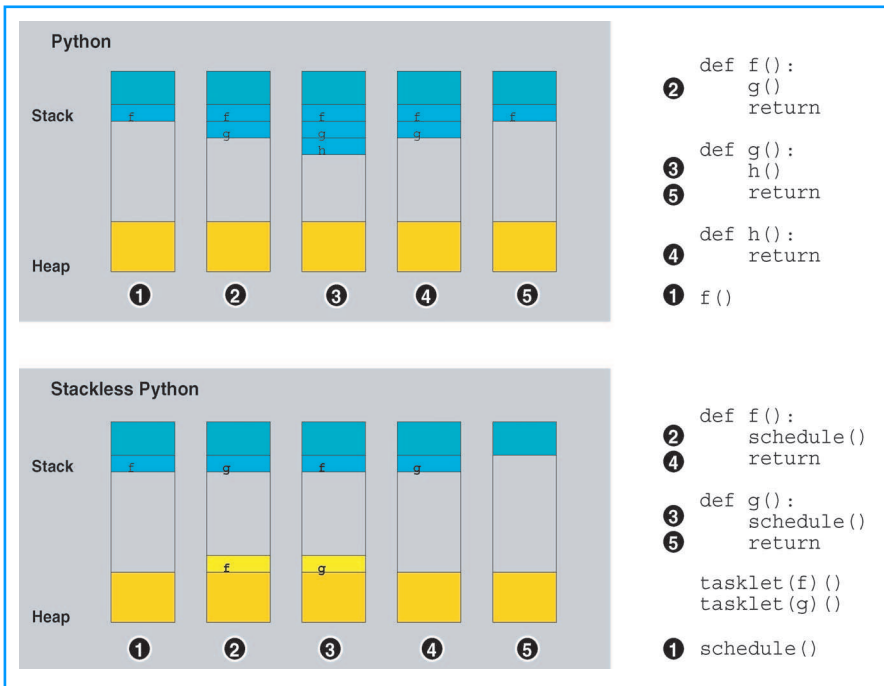


Figura 1: CPython almacena los datos pertenecientes a una subrutina en la pila sobre la función inmediatamente superior a la subrutina en la estructura jerárquica. Stackless Python cambia las Tasklets a la memoria dinámica (heap).

datos guardados en la memoria dinámica, independientemente del orden en que lleguen éstos. La Figura 1 muestra el ahorro de memoria que se obtiene mediante este procedimiento, especialmente con funciones paralelas.

Esta arquitectura permite a los programadores usar funciones como corrutinas [6], las cuales se caracterizan por las relaciones a pares en que coexisten. Pueden correr varias miles de tasklets simultáneamente, no como los hilos heredados propor-

cionados por el módulo Threading. Las Tasklets se consideran ligeras porque las podemos alternar cientos de veces por segundo. Si tenemos una tarea que depende de esta capacidad, obtendremos una implementación más rápida mediante stackless que usando hilos. El juego de rol en línea Eve Online [7] es un buen ejemplo de esto.

No es casualidad que CCP [8], la gente que está detrás de Eve Online, ayude a mantener actualizado Stackless Python. Además de CCP, Ironport [9] también usa Stackless para sus dispositivos de seguridad para redes

Ceñirse al Plan

Stackless cuenta con un programador de tareas cooperativo que usa un método de todos-contra-todos [10], es decir, que permite a cada tasklet ejecutarse en sucesión por un instante. Aunque a los programadores les parezca que las Tasklets corren en realidad de forma paralela, no es del todo cierto; dicho de otro modo, se está intentando ampliar Stackless Python para que soporte la ejecución paralela. A día de hoy los programadores han de vivir con el hecho de que el programa entero se cuelgue si se congela una Tasklet. Siempre que sea posible, es esencial evitar las llamadas al sistema que ralentizan una

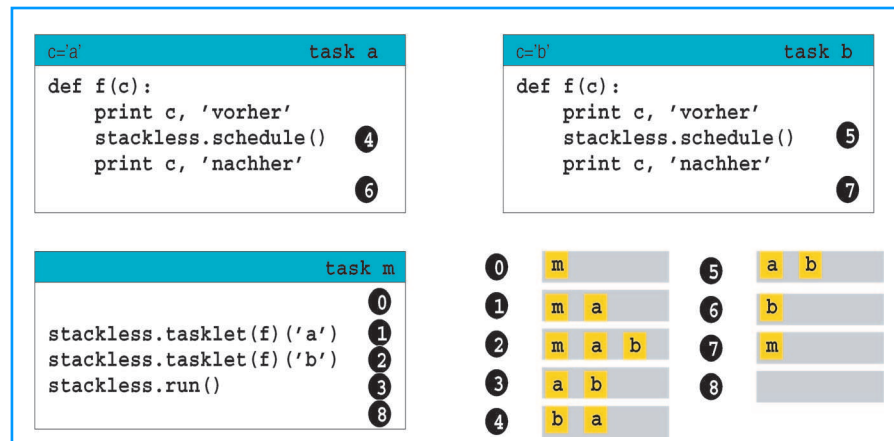


Figura 2: Orden de ejecución del programa del Listado 2.

```

Listado 2: example.py
01 import stackless
02
03 def f(c):
04   print c, 'before'
05   stackless.schedule()
06   print c, 'after'
07
08 stackless.tasklet(f)('a')
09 stackless.tasklet(f)('b')
10 stackless.run()
                
```

aplicación, como las conexiones a bases de datos o comunicaciones en red.

El programa principal aprovecha el comportamiento cooperativo de las Tasklets. Tan pronto como el programador de tareas llama a una Tasklet, ésta toma el control completo del flujo del programa. La Tasklet puede devolver el control al programa utilizando uno de las siguientes dos maneras:

- Hacer una llamada a `stackless.schedule()`
- Leer o escribir en un canal

La función `stackless.tasklet(function)()` inicializa una Tasklet y la activa en el programador de tareas. El Listado 1 muestra los primeros pasos en el intérprete interactivo. Este código nos da los mismos resultados:

```
t = stackless.tasklet()(t.
bind(f)
```

```
# pasar los parámetros de
# inicio de f:
t.setup()
# Añadir t al listado de
# tareas:
t.insert()
```

Cuando se usan hilos y herencia, el programa principal es también un hilo. De igual modo, Stackless tiene una Tasklet principal.

En el Listado 1, la Tasklet principal entrega el control a `f_task` la primera vez que llama a `stackless.schedule()`. La llamada de `f_task` a `schedule()` se lo devuelve a la consola interactiva. La consola espera entonces a que se

Listado 3: sort.py

```
01 import stackless 13
02 import random 14 ch = stackless.channel()
03 15 for each in numbers:
04 numbers = range(20) 16 stackless.tasklet
05 random.shuffle(numbers) 17 (counter)(each, ch)
06 print numbers 18 stackless.run()
07 print 'Sorting...' 19 rlist = []
08 20 while ch.balance:
09 def counter(n, ch): 21 rlist.append(ch.receive())
10 for i in xrange(n):
11 stackless.schedule()
12 ch.send(n)
```

Listado 4: Los Elementos de network_simulation.py

```
01 class Element: 07 while True:
02 def __init__(self, channel): 08 message =
03 stackless.tasklet 09 self.channel.receive()
(self.taskloop)(channel) 10 # hacer algo con el
04 self.channel = channel 11 mensaje
05 12 [...]
06 def taskloop(self):
```

Listado 5: Clase HUB en network_simulation.py

```
01 class HUB(Actor): 13 self.connectors[:]
02 def __init__(self, name, 14 while conn:
in_channel): 15 out = conn.pop()
03 Actor.__init__(self, name, 16 if out.balance < 0:
in_channel) 17 out.send(msg)
04 self.connectors = [] 18 else:
05 self.messages = [] 19 conn.insert(0,out)
06 20 if
07 def action(self, msg): 21 self.in_channel.balance > 0:
08 # consignar el paquete 22 self.messages.append(self.in_c
entrante a todos los 23 hannel.receive())
dispositivos conectados 24
09 self.messages.append(msg) 25
10 while self.messages: 26
11 msg = 27 self.messages.append(self.in_c
self.messages.pop() 28 hannel.receive())
12 conn = 29
30 stackless.schedule()
```

Listado 6: squareroot.py

```
01 import stackless
02 import pickle
03
04 def squareroot(x):
05 # Método Newton
06 print
07 print "Raíz cuadrada de ",x
08 print "-----"
09 i=0
10 y=x
11 print i, ":", y
12 while True:
13 i += 1
14
15 y = (y + x/y)/2.0
16 print i, ":", y
17 stackless.schedule()
18
19 if __name__ == '__main__':
20 import sys
21 x = float(sys.argv[5])
22 task =
stackless.tasklet(squareroot)
(x)
23 stackless.schedule()
24 stackless.schedule()
25 print 'pickle'
26 pickled_task =
pickle.dumps(task)
27 task.remove()
28 print 'unpickle'
29 newtask =
pickle.loads(pickled_task)
30 newtask.insert()
31 stackless.schedule()
```

produzca una entrada, por lo que inicialmente no sucede nada. El programador de tareas permite entonces que *f_task* ejecute su comando final.

El Listado 2 muestra un programa simple que no hace mucho más que el ejemplo anterior. Crea dos Tasklets, pero esta vez las inicia llamando a *stackless.run()*.

Al contrario que *stackless.schedule()*, se elimina del programador de tareas la Tasklet invocada (en este caso la principal). La llamada termina cuando el programador de tareas se queda sin Tasklets y pasa el turno al proceso padre (Figura 2). Como cualquier otro ejemplo de este artículo, el programa mostrado en el Listado 2 se puede descargar del sitio web de Linux Magazine [11].

Si el programa principal del Listado 2 usase *schedule()* en lugar de *run()*, los resultados serían distintos; ninguna de las Tasklets llegaría siquiera a la línea *print c*, 'after' de la función *f(c)*, ya que el programa principal terminaría antes de ejecutar las Tasklets restantes. El Listado 3, aunque con un algoritmo sin mucha utilidad, nos muestra el uso de canales. Como puede apreciarse, el flujo del programa es determinante con Stackless, a diferencia de lo que ocurre con los hilos de los sistemas operativos.

Sólo una tasklet se ejecuta cada vez, lo que significa que el procesamiento paralelo genuino no se da ni en sistemas multiprocesador.

Además de esto, las Tasklets deciden por sí mismas cuándo pasan el control del flujo del programa; de este modo no hay factores externos

que no pueda manejar el programa.

network_simulation.py implementa la simulación de una red simple. Los elementos principales son nodos que representan a estaciones conectadas y hubs que conectan las estaciones. Un nodo recibe paquetes, reenviando todos los que no van dirigidos a él. Y un hub reenvía todos los paquetes al resto de dispositivos conectados a la red.

Para simplificar las cosas, hay un solo canal de recepción para todos los objetos. El Listado 4 muestra la estructura básica de los nodos y hubs.

La simulación es la misma tanto si tenemos 10 como si son 1000 máquinas. Cada elemento actúa de forma independiente al resto de elementos, con unas comunicaciones basadas simplemente en los mensajes que intercambian.

Al diseñar el flujo del programa debemos evitar que las Tasklets se bloqueen las unas a las otras. En nuestro ejemplo, cada elemento individual bloquea el flujo del programa hasta que un mensaje alcanza el canal de control.

Las Tasklets también paran cuando tienen algo que enviar. Esto hace que el bucle del programa para el hub sea algo más complejo (ver el Listado 5).

El hub sólo envía un mensaje cuando un nodo o un hub está esperando al otro lado de la línea. La petición *out.balance < 0* es la encargada de ello. Adicionalmente, el hub tiene que recoger los paquetes entrantes. Si el atributo de balance de un canal es positivo, otro nodo de

la red comienza a esperar para enviar algo.

En Conserva

squareroot.py, en el Listado 6, muestra una característica menos conocida de Stackless; podemos almacenar Tasklets en formato binario (llamadas 'conservas' en el mundillo de Python). Y también guardar el estado interno de una Tasklet para restaurarla más adelante. El programa de muestra usa el método Newton para calcular la raíz cuadrada de un número. *squareroot.py* genera la salida mostrada más abajo pasándole 2 como único parámetro:

```
01 $ stackless squareroot2
    .py 2
02 Raíz Cuadrada de 2.0
03 - - - - -
04 0 : 2.0
05 1 : 1.5
06 2 : 1.41666666667
07 pickle
08 unpickle
09 3 : 1.41421568627
10 4 : 1.41421356237
```

Después de almacenar la tasklet, podemos ejecutarla desde otro proceso, en otra máquina, incluso posiblemente en otra arquitectura. ■

Tabla 1: Clases y Funciones en Stackless

<code>task=stackless.tasklet (<función>)</code> (<argumentos de la función>)	Crea una tarea Tasklet
<code>stackless.schedule()</code>	Cambia a la siguiente Tasklet
<code>stackless.run()</code>	Cambia a la siguiente Tasklet y se autoelimina del listado de tareas
<code>channel=stackless.channel()</code>	Crea <i>channel</i> , un nuevo objeto de canal
<code>channel.send(message)</code>	Envía el mensaje al canal; la Tasklet lo bloquea hasta que éste ha sido recogido
<code>message=channel.receive()</code>	Recibe el mensaje desde el canal; la Tasklet lo bloquea hasta que éste llega
<code>channel.balance</code>	> 0: Alguien está esperando para enviar < 0: Alguien está esperando para recibir

RECURSOS

- [1] Stackless Python: <http://www.stackless.com>
- [2] Erlang: <http://www.erlang.org>
- [3] Oz: <http://www.mozart-oz.org>
- [4] Stack: [http://es.wikipedia.org/wiki/Pila_\(estructura_de_datos\)](http://es.wikipedia.org/wiki/Pila_(estructura_de_datos))
- [5] Heap: <http://es.wikipedia.org/wiki/Heap>
- [6] Coroutine (corrutina): <http://en.wikipedia.org/wiki/Coroutine>
- [7] Eve Online: <http://www.eve-online.com>
- [8] CCP: <http://www.ccpgames.com>
- [9] Ironport: <http://www.ironport.com>
- [10] Método todos-contra-todos: http://en.wikipedia.org/wiki/Round_robin
- [11] Listings: <http://www.linux-magazine.es/Magazine/Downloads/32>
- [12] Pypy: <http://codespeak.net/pypy/dist/pypy/doc/news.html>