

Programming with Stackless Python

IN THE HEAP

The Stackless extension brings lightweight processes to Python, opening a new style of programming with dynamic heap access.

BY STEPHAN DIEHL

Stackless Python [1] by Christian Tismer extends the popular Python interpreter, adding elements that facilitate the development of scalable applications. Small, independent program sections are encapsulated in *tasklets*. These tasklets use channels to communicate in an approach reminiscent of the Erlang [2] and Oz [3] languages. The name *Stackless* refers to the encapsulated functions that tasklets

allow to be swapped out from stack [4] to heap [5] (dynamic) memory. Data stored at this location can be accessed at any time, regardless of the order in which it arrived. Figure 1 shows how this approach can save memory, especially with parallel functions.

This architecture gives programmers the ability to use functions as coroutines [6]. Coroutines are characterized by the peer relationships in which they coexist.

Listing 1: First Steps

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

In contrast to the legacy threads provided by the *Threading* module, several thousand tasklets can run simultaneously. Tasklets are considered lightweight because they can be switched several hundred thousand times per second. If you have a task that relies on this ability, a stackless implementation will

Installation

As of this writing, no Stackless Python binary packages are available for any distribution. You will need to use Subversion to download the current release; the Stackless homepage points to the current address. Versions for Python 2.4 and 2.5 are available. After downloading, just follow the normal steps to build and install:

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

The softlink avoids possible conflicts with your existing Python installation. For more details, refer to the Stackless documentation on the project homepage.

be far faster than a threaded version. The online role-playing game, Eve Online [7], is a good example of this.

It is no coincidence that CCP [8], the people behind Eve Online, help to keep Stackless Python up to date. Apart from CCP, Ironport [9] also uses Stackless for its network security appliances.

Keeping to the Schedule

Stackless relies on a cooperative scheduler that uses a round-robin approach [10]; that is, it lets every tasklet run in succession for a short while. Although programmers can pretend that the tasklets are actually running parallel to one another, this is not strictly true; this said, there are moves to extend Stackless Python to support parallel execution. For the time being, programmers have to live with the fact that the whole program freezes if a tasklet freezes. Wherever possible, it is critical to avoid system calls that slow down an application such as network or database connections.

The main program relies on the tasklets' cooperative behavior; as soon as a tasklet is called by the scheduler, it has full control of the program flow. The tasklet has two options for returning control to the program:

- making a call to `stackless.schedule()`;
 - reading or writing on a channel.
- `stackless.tasklet(function)()` initializes a tasklet and activates it in the scheduler. Listing 1 shows the first steps in the interactive interpreter. This slightly convoluted code gives you the same results:

```
t = stackless.tasklet()
t.bind(f)
# pass in f start parameters:
t.setup()
# Append t to scheduler list:
t.insert()
```

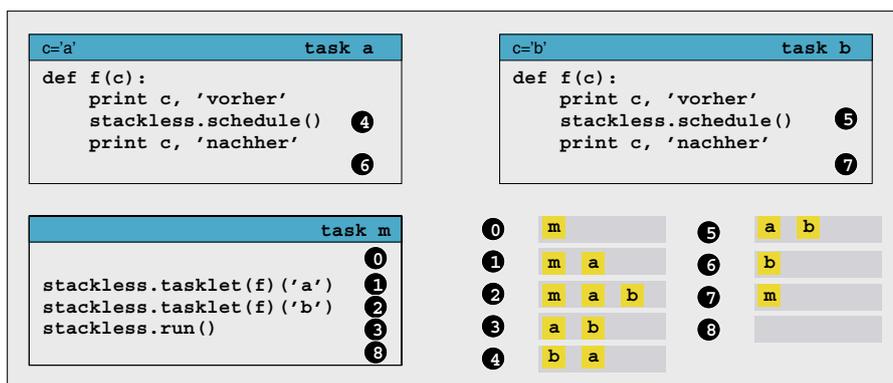


Figure 2: The order in which the program from Listing 2 is executed.

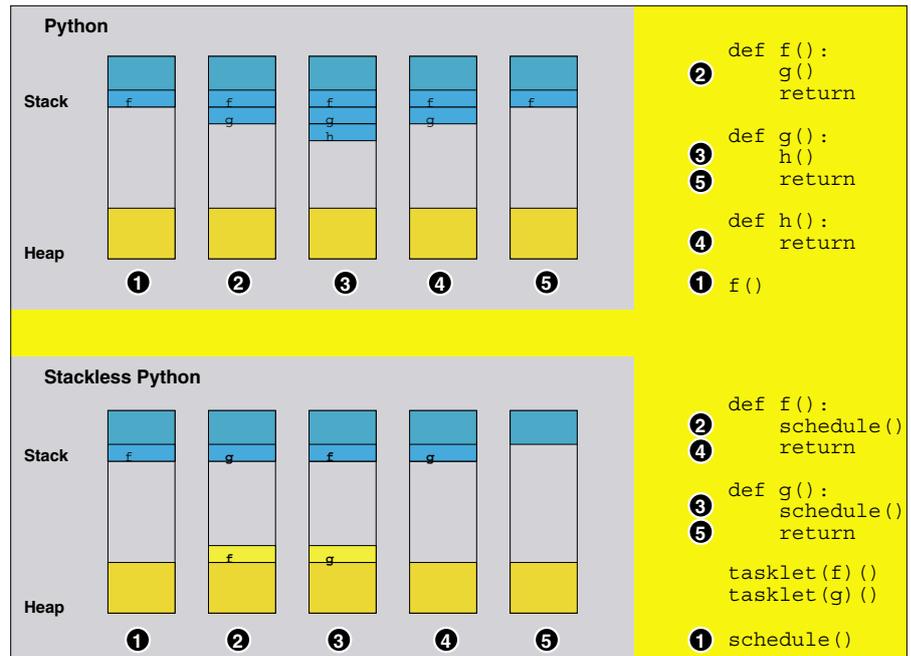


Figure 1: CPython stores the data belonging to a subroutine in stack memory above the function that is superordinate to the subroutine in the hierarchical structure. Stackless Python swaps tasklets out to heap memory.

In legacy threading, the main program is also a thread. In line with this, Stackless has a main tasklet.

In Listing 1, the main tasklet hands control over to `f_task` the first time it calls `stackless.schedule()`. `f_task`'s call to `schedule()` then hands control back to the interactive console. The console waits for input at this point, so nothing happens initially. The scheduler then lets `f_task` run its final command.

Listing 2 shows a simple program that does not really do much more than the previous example. It creates two tasklets, but this time it starts processing them by calling `stackless.run()`.

In contrast to `stackless.schedule()`, the tasklet that is called – the main tasklet in this case – is removed from the scheduler. The call finishes when the sched-

uler runs out of tasklets, and the parent process is then scheduled (Figure 2). Like all the other examples in this article, the program shown in Listing 2 is available for download at the *Linux Magazine* website [11].

If the main program of Listing 2 were to use `schedule()` instead of `run()`, the results would be different; neither tasklet would even reach the `print c, 'after'` line in `f()` because the main program would exit before running the outstanding tasklets. Listing 3 demonstrates the use of channels with an example of a fairly inefficient sort algorithm. As you can see, Stackless' program flow is deterministic, in contrast to operating system threads.

Only one tasklet runs at any time, which means that genuine parallel pro-

Listing 2: example.p

```
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()
```

cessing does not occur even on multiprocessor systems.

Besides this, tasklets decide themselves when they hand over control of program flow; thus, there are no external factors that the program cannot handle.

`network_simulation.py` implements a simple network simulation. The main elements are nodes that represent network-attached computers and hubs that connect the computers. A node receives packets and forwards any packets not addressed to it.

A hub receives packets and forwards them to all the devices connected to the network.

To keep things simple, all the objects have just one reception channel. Listing 4 shows the basic structure of the nodes and hubs.

Whether you have 10 or 1000 computers, it makes no difference in the simulation. Each element acts independently of all other elements, with communications simply relying on the messages that they exchange.

When designing program flow, it is important to avoid tasklets blocking each other. In our example, the individual elements block the program flow until a message reaches the control channel.

The tasklets also take a break when they have something to send. This makes the program loop for the hub slightly more complex (see Listing 5).

The hub will only send a message when a node or hub is listening at the other end of the line. The `out.balance < 0` request takes care of this. Additionally, the hub needs to pick up incoming pack-

Listing 3: sort.py

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

Listing 4: The Elements in network_simulation.py

```
01 class Element:
02     def __init__(self, channel):
03         stackless.tasklet(self.
04         taskloop)(channel)
05         self.channel = channel
06
07     def taskloop(self):
08         while True:
09             message = self.channel.
10             receive()
11             # do something with the
12             message
13             [...]
14
15     while True:
16         message = self.channel.
17         receive()
18         # do something with the
19         message
20         [...]
21
22     while True:
23         message = self.channel.
24         receive()
25         # do something with the
26         message
27         [...]
28
29     while True:
30         message = self.channel.
31         receive()
32         # do something with the
33         message
34         [...]
```

Listing 5: Class HUB in network_simulation.py

```
01 class HUB(Actor):
02     def __init__(self, name, in_
03     channel):
04         Actor.__init__(self, name,
05         in_channel)
06         self.connectors = []
07         self.messages = []
08
09     def action(self, msg):
10         # dispatch incoming packet
11         to all connected devices
12         self.messages.append(msg)
13         while self.messages:
14             msg = self.messages.
15             pop()
16             conn = self.connectors[:]
17             while conn:
18                 out = conn.pop()
19                 if out.balance < 0:
20                     out.send(msg)
21                 else:
22                     conn.insert(0,out)
23                     if self.in_channel.
24                     balance > 0:
25                         self.messages.
26                         append(self.in_channel.
27                         receive())
28                         stackless.schedule()
29
30     def __init__(self, name, in_
31     channel):
32         Actor.__init__(self, name,
33         in_channel)
34         self.connectors = []
35         self.messages = []
36
37     def action(self, msg):
38         # dispatch incoming packet
39         to all connected devices
40         self.messages.append(msg)
41         while self.messages:
42             msg = self.messages.
43             pop()
44             conn = self.connectors[:]
45             while conn:
46                 out = conn.pop()
47                 if out.balance < 0:
48                     out.send(msg)
49                 else:
50                     conn.insert(0,out)
51                     if self.in_channel.
52                     balance > 0:
53                         self.messages.
54                         append(self.in_channel.
55                         receive())
56                         stackless.schedule()
57
58     def __init__(self, name, in_
59     channel):
60         Actor.__init__(self, name,
61         in_channel)
62         self.connectors = []
63         self.messages = []
64
65     def action(self, msg):
66         # dispatch incoming packet
67         to all connected devices
68         self.messages.append(msg)
69         while self.messages:
70             msg = self.messages.
71             pop()
72             conn = self.connectors[:]
73             while conn:
74                 out = conn.pop()
75                 if out.balance < 0:
76                     out.send(msg)
77                 else:
78                     conn.insert(0,out)
79                     if self.in_channel.
80                     balance > 0:
81                         self.messages.
82                         append(self.in_channel.
83                         receive())
84                         stackless.schedule()
85
86     def __init__(self, name, in_
87     channel):
88         Actor.__init__(self, name,
89         in_channel)
90         self.connectors = []
91         self.messages = []
92
93     def action(self, msg):
94         # dispatch incoming packet
95         to all connected devices
96         self.messages.append(msg)
97         while self.messages:
98             msg = self.messages.
99             pop()
100            conn = self.connectors[:]
101            while conn:
102                out = conn.pop()
103                if out.balance < 0:
104                    out.send(msg)
105                else:
106                    conn.insert(0,out)
107                    if self.in_channel.
108                    balance > 0:
109                        self.messages.
110                        append(self.in_channel.
111                        receive())
112                        stackless.schedule()
113
114     def __init__(self, name, in_
115     channel):
116         Actor.__init__(self, name,
117         in_channel)
118         self.connectors = []
119         self.messages = []
120
121     def action(self, msg):
122         # dispatch incoming packet
123         to all connected devices
124         self.messages.append(msg)
125         while self.messages:
126             msg = self.messages.
127             pop()
128             conn = self.connectors[:]
129             while conn:
130                 out = conn.pop()
131                 if out.balance < 0:
132                     out.send(msg)
133                 else:
134                     conn.insert(0,out)
135                     if self.in_channel.
136                     balance > 0:
137                         self.messages.
138                         append(self.in_channel.
139                         receive())
140                         stackless.schedule()
141
142     def __init__(self, name, in_
143     channel):
144         Actor.__init__(self, name,
145         in_channel)
146         self.connectors = []
147         self.messages = []
148
149     def action(self, msg):
150         # dispatch incoming packet
151         to all connected devices
152         self.messages.append(msg)
153         while self.messages:
154             msg = self.messages.
155             pop()
156             conn = self.connectors[:]
157             while conn:
158                 out = conn.pop()
159                 if out.balance < 0:
160                     out.send(msg)
161                 else:
162                     conn.insert(0,out)
163                     if self.in_channel.
164                     balance > 0:
165                         self.messages.
166                         append(self.in_channel.
167                         receive())
168                         stackless.schedule()
169
170     def __init__(self, name, in_
171     channel):
172         Actor.__init__(self, name,
173         in_channel)
174         self.connectors = []
175         self.messages = []
176
177     def action(self, msg):
178         # dispatch incoming packet
179         to all connected devices
180         self.messages.append(msg)
181         while self.messages:
182             msg = self.messages.
183             pop()
184             conn = self.connectors[:]
185             while conn:
186                 out = conn.pop()
187                 if out.balance < 0:
188                     out.send(msg)
189                 else:
190                     conn.insert(0,out)
191                     if self.in_channel.
192                     balance > 0:
193                         self.messages.
194                         append(self.in_channel.
195                         receive())
196                         stackless.schedule()
197
198     def __init__(self, name, in_
199     channel):
200         Actor.__init__(self, name,
201         in_channel)
202         self.connectors = []
203         self.messages = []
204
205     def action(self, msg):
206         # dispatch incoming packet
207         to all connected devices
208         self.messages.append(msg)
209         while self.messages:
210             msg = self.messages.
211             pop()
212             conn = self.connectors[:]
213             while conn:
214                 out = conn.pop()
215                 if out.balance < 0:
216                     out.send(msg)
217                 else:
218                     conn.insert(0,out)
219                     if self.in_channel.
220                     balance > 0:
221                         self.messages.
222                         append(self.in_channel.
223                         receive())
224                         stackless.schedule()
225
226     def __init__(self, name, in_
227     channel):
228         Actor.__init__(self, name,
229         in_channel)
230         self.connectors = []
231         self.messages = []
232
233     def action(self, msg):
234         # dispatch incoming packet
235         to all connected devices
236         self.messages.append(msg)
237         while self.messages:
238             msg = self.messages.
239             pop()
240             conn = self.connectors[:]
241             while conn:
242                 out = conn.pop()
243                 if out.balance < 0:
244                     out.send(msg)
245                 else:
246                     conn.insert(0,out)
247                     if self.in_channel.
248                     balance > 0:
249                         self.messages.
250                         append(self.in_channel.
251                         receive())
252                         stackless.schedule()
253
254     def __init__(self, name, in_
255     channel):
256         Actor.__init__(self, name,
257         in_channel)
258         self.connectors = []
259         self.messages = []
260
261     def action(self, msg):
262         # dispatch incoming packet
263         to all connected devices
264         self.messages.append(msg)
265         while self.messages:
266             msg = self.messages.
267             pop()
268             conn = self.connectors[:]
269             while conn:
270                 out = conn.pop()
271                 if out.balance < 0:
272                     out.send(msg)
273                 else:
274                     conn.insert(0,out)
275                     if self.in_channel.
276                     balance > 0:
277                         self.messages.
278                         append(self.in_channel.
279                         receive())
280                         stackless.schedule()
281
282     def __init__(self, name, in_
283     channel):
284         Actor.__init__(self, name,
285         in_channel)
286         self.connectors = []
287         self.messages = []
288
289     def action(self, msg):
290         # dispatch incoming packet
291         to all connected devices
292         self.messages.append(msg)
293         while self.messages:
294             msg = self.messages.
295             pop()
296             conn = self.connectors[:]
297             while conn:
298                 out = conn.pop()
299                 if out.balance < 0:
300                     out.send(msg)
301                 else:
302                     conn.insert(0,out)
303                     if self.in_channel.
304                     balance > 0:
305                         self.messages.
306                         append(self.in_channel.
307                         receive())
308                         stackless.schedule()
309
310     def __init__(self, name, in_
311     channel):
312         Actor.__init__(self, name,
313         in_channel)
314         self.connectors = []
315         self.messages = []
316
317     def action(self, msg):
318         # dispatch incoming packet
319         to all connected devices
320         self.messages.append(msg)
321         while self.messages:
322             msg = self.messages.
323             pop()
324             conn = self.connectors[:]
325             while conn:
326                 out = conn.pop()
327                 if out.balance < 0:
328                     out.send(msg)
329                 else:
330                     conn.insert(0,out)
331                     if self.in_channel.
332                     balance > 0:
333                         self.messages.
334                         append(self.in_channel.
335                         receive())
336                         stackless.schedule()
337
338     def __init__(self, name, in_
339     channel):
340         Actor.__init__(self, name,
341         in_channel)
342         self.connectors = []
343         self.messages = []
344
345     def action(self, msg):
346         # dispatch incoming packet
347         to all connected devices
348         self.messages.append(msg)
349         while self.messages:
350             msg = self.messages.
351             pop()
352             conn = self.connectors[:]
353             while conn:
354                 out = conn.pop()
355                 if out.balance < 0:
356                     out.send(msg)
357                 else:
358                     conn.insert(0,out)
359                     if self.in_channel.
360                     balance > 0:
361                         self.messages.
362                         append(self.in_channel.
363                         receive())
364                         stackless.schedule()
365
366     def __init__(self, name, in_
367     channel):
368         Actor.__init__(self, name,
369         in_channel)
370         self.connectors = []
371         self.messages = []
372
373     def action(self, msg):
374         # dispatch incoming packet
375         to all connected devices
376         self.messages.append(msg)
377         while self.messages:
378             msg = self.messages.
379             pop()
380             conn = self.connectors[:]
381             while conn:
382                 out = conn.pop()
383                 if out.balance < 0:
384                     out.send(msg)
385                 else:
386                     conn.insert(0,out)
387                     if self.in_channel.
388                     balance > 0:
389                         self.messages.
390                         append(self.in_channel.
391                         receive())
392                         stackless.schedule()
393
394     def __init__(self, name, in_
395     channel):
396         Actor.__init__(self, name,
397         in_channel)
398         self.connectors = []
399         self.messages = []
400
401     def action(self, msg):
402         # dispatch incoming packet
403         to all connected devices
404         self.messages.append(msg)
405         while self.messages:
406             msg = self.messages.
407             pop()
408             conn = self.connectors[:]
409             while conn:
410                 out = conn.pop()
411                 if out.balance < 0:
412                     out.send(msg)
413                 else:
414                     conn.insert(0,out)
415                     if self.in_channel.
416                     balance > 0:
417                         self.messages.
418                         append(self.in_channel.
419                         receive())
420                         stackless.schedule()
421
422     def __init__(self, name, in_
423     channel):
424         Actor.__init__(self, name,
425         in_channel)
426         self.connectors = []
427         self.messages = []
428
429     def action(self, msg):
430         # dispatch incoming packet
431         to all connected devices
432         self.messages.append(msg)
433         while self.messages:
434             msg = self.messages.
435             pop()
436             conn = self.connectors[:]
437             while conn:
438                 out = conn.pop()
439                 if out.balance < 0:
440                     out.send(msg)
441                 else:
442                     conn.insert(0,out)
443                     if self.in_channel.
444                     balance > 0:
445                         self.messages.
446                         append(self.in_channel.
447                         receive())
448                         stackless.schedule()
449
450     def __init__(self, name, in_
451     channel):
452         Actor.__init__(self, name,
453         in_channel)
454         self.connectors = []
455         self.messages = []
456
457     def action(self, msg):
458         # dispatch incoming packet
459         to all connected devices
460         self.messages.append(msg)
461         while self.messages:
462             msg = self.messages.
463             pop()
464             conn = self.connectors[:]
465             while conn:
466                 out = conn.pop()
467                 if out.balance < 0:
468                     out.send(msg)
469                 else:
470                     conn.insert(0,out)
471                     if self.in_channel.
472                     balance > 0:
473                         self.messages.
474                         append(self.in_channel.
475                         receive())
476                         stackless.schedule()
477
478     def __init__(self, name, in_
479     channel):
480         Actor.__init__(self, name,
481         in_channel)
482         self.connectors = []
483         self.messages = []
484
485     def action(self, msg):
486         # dispatch incoming packet
487         to all connected devices
488         self.messages.append(msg)
489         while self.messages:
490             msg = self.messages.
491             pop()
492             conn = self.connectors[:]
493             while conn:
494                 out = conn.pop()
495                 if out.balance < 0:
496                     out.send(msg)
497                 else:
498                     conn.insert(0,out)
499                     if self.in_channel.
500                     balance > 0:
501                         self.messages.
502                         append(self.in_channel.
503                         receive())
504                         stackless.schedule()
505
506     def __init__(self, name, in_
507     channel):
508         Actor.__init__(self, name,
509         in_channel)
510         self.connectors = []
511         self.messages = []
512
513     def action(self, msg):
514         # dispatch incoming packet
515         to all connected devices
516         self.messages.append(msg)
517         while self.messages:
518             msg = self.messages.
519             pop()
520             conn = self.connectors[:]
521             while conn:
522                 out = conn.pop()
523                 if out.balance < 0:
524                     out.send(msg)
525                 else:
526                     conn.insert(0,out)
527                     if self.in_channel.
528                     balance > 0:
529                         self.messages.
530                         append(self.in_channel.
531                         receive())
532                         stackless.schedule()
533
534     def __init__(self, name, in_
535     channel):
536         Actor.__init__(self, name,
537         in_channel)
538         self.connectors = []
539         self.messages = []
540
541     def action(self, msg):
542         # dispatch incoming packet
543         to all connected devices
544         self.messages.append(msg)
545         while self.messages:
546             msg = self.messages.
547             pop()
548             conn = self.connectors[:]
549             while conn:
550                 out = conn.pop()
551                 if out.balance < 0:
552                     out.send(msg)
553                 else:
554                     conn.insert(0,out)
555                     if self.in_channel.
556                     balance > 0:
557                         self.messages.
558                         append(self.in_channel.
559                         receive())
560                         stackless.schedule()
561
562     def __init__(self, name, in_
563     channel):
564         Actor.__init__(self, name,
565         in_channel)
566         self.connectors = []
567         self.messages = []
568
569     def action(self, msg):
570         # dispatch incoming packet
571         to all connected devices
572         self.messages.append(msg)
573         while self.messages:
574             msg = self.messages.
575             pop()
576             conn = self.connectors[:]
577             while conn:
578                 out = conn.pop()
579                 if out.balance < 0:
580                     out.send(msg)
581                 else:
582                     conn.insert(0,out)
583                     if self.in_channel.
584                     balance > 0:
585                         self.messages.
586                         append(self.in_channel.
587                         receive())
588                         stackless.schedule()
589
590     def __init__(self, name, in_
591     channel):
592         Actor.__init__(self, name,
593         in_channel)
594         self.connectors = []
595         self.messages = []
596
597     def action(self, msg):
598         # dispatch incoming packet
599         to all connected devices
600         self.messages.append(msg)
601         while self.messages:
602             msg = self.messages.
603             pop()
604             conn = self.connectors[:]
605             while conn:
606                 out = conn.pop()
607                 if out.balance < 0:
608                     out.send(msg)
609                 else:
610                     conn.insert(0,out)
611                     if self.in_channel.
612                     balance > 0:
613                         self.messages.
614                         append(self.in_channel.
615                         receive())
616                         stackless.schedule()
617
618     def __init__(self, name, in_
619     channel):
620         Actor.__init__(self, name,
621         in_channel)
622         self.connectors = []
623         self.messages = []
624
625     def action(self, msg):
626         # dispatch incoming packet
627         to all connected devices
628         self.messages.append(msg)
629         while self.messages:
630             msg = self.messages.
631             pop()
632             conn = self.connectors[:]
633             while conn:
634                 out = conn.pop()
635                 if out.balance < 0:
636                     out.send(msg)
637                 else:
638                     conn.insert(0,out)
639                     if self.in_channel.
640                     balance > 0:
641                         self.messages.
642                         append(self.in_channel.
643                         receive())
644                         stackless.schedule()
645
646     def __init__(self, name, in_
647     channel):
648         Actor.__init__(self, name,
649         in_channel)
650         self.connectors = []
651         self.messages = []
652
653     def action(self, msg):
654         # dispatch incoming packet
655         to all connected devices
656         self.messages.append(msg)
657         while self.messages:
658             msg = self.messages.
659             pop()
660             conn = self.connectors[:]
661             while conn:
662                 out = conn.pop()
663                 if out.balance < 0:
664                     out.send(msg)
665                 else:
666                     conn.insert(0,out)
667                     if self.in_channel.
668                     balance > 0:
669                         self.messages.
670                         append(self.in_channel.
671                         receive())
672                         stackless.schedule()
673
674     def __init__(self, name, in_
675     channel):
676         Actor.__init__(self, name,
677         in_channel)
678         self.connectors = []
679         self.messages = []
680
681     def action(self, msg):
682         # dispatch incoming packet
683         to all connected devices
684         self.messages.append(msg)
685         while self.messages:
686             msg = self.messages.
687             pop()
688             conn = self.connectors[:]
689             while conn:
690                 out = conn.pop()
691                 if out.balance < 0:
692                     out.send(msg)
693                 else:
694                     conn.insert(0,out)
695                     if self.in_channel.
696                     balance > 0:
697                         self.messages.
698                         append(self.in_channel.
699                         receive())
700                         stackless.schedule()
701
702     def __init__(self, name, in_
703     channel):
704         Actor.__init__(self, name,
705         in_channel)
706         self.connectors = []
707         self.messages = []
708
709     def action(self, msg):
710         # dispatch incoming packet
711         to all connected devices
712         self.messages.append(msg)
713         while self.messages:
714             msg = self.messages.
715             pop()
716             conn = self.connectors[:]
717             while conn:
718                 out = conn.pop()
719                 if out.balance < 0:
720                     out.send(msg)
721                 else:
722                     conn.insert(0,out)
723                     if self.in_channel.
724                     balance > 0:
725                         self.messages.
726                         append(self.in_channel.
727                         receive())
728                         stackless.schedule()
729
730     def __init__(self, name, in_
731     channel):
732         Actor.__init__(self, name,
733         in_channel)
734         self.connectors = []
735         self.messages = []
736
737     def action(self, msg):
738         # dispatch incoming packet
739         to all connected devices
740         self.messages.append(msg)
741         while self.messages:
742             msg = self.messages.
743             pop()
744             conn = self.connectors[:]
745             while conn:
746                 out = conn.pop()
747                 if out.balance < 0:
748                     out.send(msg)
749                 else:
750                     conn.insert(0,out)
751                     if self.in_channel.
752                     balance > 0:
753                         self.messages.
754                         append(self.in_channel.
755                         receive())
756                         stackless.schedule()
757
758     def __init__(self, name, in_
759     channel):
760         Actor.__init__(self, name,
761         in_channel)
762         self.connectors = []
763         self.messages = []
764
765     def action(self, msg):
766         # dispatch incoming packet
767         to all connected devices
768         self.messages.append(msg)
769         while self.messages:
770             msg = self.messages.
771             pop()
772             conn = self.connectors[:]
773             while conn:
774                 out = conn.pop()
775                 if out.balance < 0:
776                     out.send(msg)
777                 else:
778                     conn.insert(0,out)
779                     if self.in_channel.
780                     balance > 0:
781                         self.messages.
782                         append(self.in_channel.
783                         receive())
784                         stackless.schedule()
785
786     def __init__(self, name, in_
787     channel):
788         Actor.__init__(self, name,
789         in_channel)
790         self.connectors = []
791         self.messages = []
792
793     def action(self, msg):
794         # dispatch incoming packet
795         to all connected devices
796         self.messages.append(msg)
797         while self.messages:
798             msg = self.messages.
799             pop()
800             conn = self.connectors[:]
801             while conn:
802                 out = conn.pop()
803                 if out.balance < 0:
804                     out.send(msg)
805                 else:
806                     conn.insert(0,out)
807                     if self.in_channel.
808                     balance > 0:
809                         self.messages.
810                         append(self.in_channel.
811                         receive())
812                         stackless.schedule()
813
814     def __init__(self, name, in_
815     channel):
816         Actor.__init__(self, name,
817         in_channel)
818         self.connectors = []
819         self.messages = []
820
821     def action(self, msg):
822         # dispatch incoming packet
823         to all connected devices
824         self.messages.append(msg)
825         while self.messages:
826             msg = self.messages.
827             pop()
828             conn = self.connectors[:]
829             while conn:
830                 out = conn.pop()
831                 if out.balance < 0:
832                     out.send(msg)
833                 else:
834                     conn.insert(0,out)
835                     if self.in_channel.
836                     balance > 0:
837                         self.messages.
838                         append(self.in_channel.
839                         receive())
840                         stackless.schedule()
841
842     def __init__(self, name, in_
843     channel):
844         Actor.__init__(self, name,
845         in_channel)
846         self.connectors = []
847         self.messages = []
848
849     def action(self, msg):
850         # dispatch incoming packet
851         to all connected devices
852         self.messages.append(msg)
853         while self.messages:
854             msg = self.messages.
855             pop()
856             conn = self.connectors[:]
857             while conn:
858                 out = conn.pop()
859                 if out.balance < 0:
860                     out.send(msg)
861                 else:
862                     conn.insert(0,out)
863                     if self.in_channel.
864                     balance > 0:
865                         self.messages.
866                         append(self.in_channel.
867                         receive())
868                         stackless.schedule()
869
870     def __init__(self, name, in_
871     channel):
872         Actor.__init__(self, name,
873         in_channel)
874         self.connectors = []
875         self.messages = []
876
877     def action(self, msg):
878         # dispatch incoming packet
879         to all connected devices
880         self.messages.append(msg)
881         while self.messages:
882             msg = self.messages.
883             pop()
884             conn = self.connectors[:]
885             while conn:
886                 out = conn.pop()
887                 if out.balance < 0:
888                     out.send(msg)
889                 else:
890                     conn.insert(0,out)
891                     if self.in_channel.
892                     balance > 0:
893                         self.messages.
894                         append(self.in_channel.
895                         receive())
896                         stackless.schedule()
897
898     def __init__(self, name, in_
899     channel):
900         Actor.__init__(self, name,
901         in_channel)
902         self.connectors = []
903         self.messages = []
904
905     def action(self, msg):
906         # dispatch incoming packet
907         to all connected devices
908         self.messages.append(msg)
909         while self.messages:
910             msg = self.messages.
911             pop()
912             conn = self.connectors[:]
913             while conn:
914                 out = conn.pop()
915                 if out.balance < 0:
916                     out.send(msg)
917                 else:
918                     conn.insert(0,out)
919                     if self.in_channel.
920                     balance > 0:
921                         self.messages.
922                         append(self.in_channel.
923                         receive())
924                         stackless.schedule()
925
926     def __init__(self, name, in_
927     channel):
928         Actor.__init__(self, name,
929         in_channel)
930         self.connectors = []
931         self.messages = []
932
933     def action(self, msg):
934         # dispatch incoming packet
935         to all connected devices
936         self.messages.append(msg)
937         while self.messages:
938             msg = self.messages.
939             pop()
940             conn = self.connectors[:]
941             while conn:
942                 out = conn.pop()
943                 if out.balance < 0:
944                     out.send(msg)
945                 else:
946                     conn.insert(0,out)
947                     if self.in_channel.
948                     balance > 0:
949                         self.messages.
950                         append(self.in_channel.
951                         receive())
952                         stackless.schedule()
953
954     def __init__(self, name, in_
955     channel):
956         Actor.__init__(self, name,
957         in_channel)
958         self.connectors = []
959         self.messages = []
960
961     def action(self, msg):
962         # dispatch incoming packet
963         to all connected devices
964         self.messages.append(msg)
965         while self.messages:
966             msg = self.messages.
967             pop()
968             conn = self.connectors[:]
969             while conn:
970                 out = conn.pop()
971                 if out.balance < 0:
972                     out.send(msg)
973                 else:
974                     conn.insert(0,out)
975                     if self.in_channel.
976                     balance > 0:
977                         self.messages.
978                         append(self.in_channel.
979                         receive())
980                         stackless.schedule()
981
982     def __init__(self, name, in_
983     channel):
984         Actor.__init__(self, name,
985         in_channel)
986         self.connectors = []
987         self.messages = []
988
989     def action(self, msg):
990         # dispatch incoming packet
991         to all connected devices
992         self.messages.append(msg)
993         while self.messages:
994             msg = self.messages.
995             pop()
996             conn = self.connectors[:]
997             while conn:
998                 out = conn.pop()
999                 if out.balance < 0:
1000                    out.send(msg)
1001                else:
1002                    conn.insert(0,out)
1003                    if self.in_channel.
1004                    balance > 0:
1005                        self.messages.
1006                        append(self.in_channel.
1007                        receive())
1008                        stackless.schedule()
1009
1010     def __init__(self, name, in_
1011     channel):
1012         Actor.__init__(self, name,
1013         in_channel)
1014         self.connectors = []
1015         self.messages = []
1016
1017     def action(self, msg):
1018         # dispatch incoming packet
1019         to all connected devices
1020         self.messages.append(msg)
1021         while self.messages:
1022             msg = self.messages.
1023             pop()
1024             conn = self.connectors[:]
1025             while conn:
1026                 out = conn.pop()
1027                 if out.balance < 0:
1028                     out.send(msg)
1029                 else:
1030                     conn.insert(0,out)
1031                     if self.in_channel.
1032                     balance > 0:
1033                         self.messages.
1034                         append(self.in_channel.
1035                         receive())
1036                         stackless.schedule()
1037
1038     def __init__(self, name, in_
1039     channel):
1040         Actor.__init__(self, name,
1041         in_channel)
1042         self.connectors = []
1043         self.messages = []
1044
1045     def action(self, msg):
1046         # dispatch incoming packet
1047         to all connected devices
1048         self.messages.append(msg)
1049         while self.messages:
1050             msg = self.messages.
1051             pop()
1052             conn = self.connectors[:]
1053             while conn:
1054                 out = conn.pop()
1055                 if out.balance < 0:
1056                     out.send(msg)
1057                 else:
1058                     conn.insert(0,out)
1059                     if self.in_channel.
1060                     balance > 0:
1061                         self.messages.
1062                         append(self.in_channel.
1063                         receive())
1064                         stackless.schedule()
1065
1066     def __init__(self, name, in_
1067     channel):
1068         Actor.__init__(self, name,
1069         in_channel)
1070         self.connectors = []
1071         self.messages = []
1072
1073     def action(self, msg):
1074         # dispatch incoming packet
1075         to all connected devices
1076         self.messages.append(msg)
1077         while self.messages:
1078             msg = self.messages.
1079             pop()
1080             conn = self.connectors[:]
1081             while conn:
1082                 out = conn.pop()
1083                 if out.balance < 0:
1084                     out.send(msg)
1085                 else:
1086                     conn.insert(0,out)
1087                     if self.in_channel.
1088                     balance > 0:
1089                         self.messages.
1090                         append(self.in_channel.
1091                         receive())
1092                         stackless.schedule()
1093
1094     def __init__(self, name, in_
1095     channel):
1096         Actor.__init__(self, name,
1097         in_channel)
1098         self.connectors = []
1099         self.messages = []
1100
1101     def action(self, msg):
1102         # dispatch incoming packet
1103         to all connected devices
1104         self.messages.append(msg)
1105         while self.messages:
1106             msg = self.messages.
1107             pop()
1108             conn = self.connectors[:]
1109             while conn:
1110                 out = conn.pop()
1111                 if out.balance < 0:
1112                     out.send(msg)
1113                 else:
1114                     conn.insert(0,out)
1115                     if self.in_channel.
1116                     balance > 0:
1117                         self.messages.
1118                         append(self.in_channel.
1119                         receive())
1120                         stackless.schedule()
1121
1122     def __init__(self, name, in_
1123     channel):
1124         Actor.__init__(self, name,
1125         in_channel)
1126         self.connectors = []
1127         self.messages = []
1128
1129     def action(self, msg):
1130         # dispatch incoming packet
1131         to all connected devices
1132         self.messages.append(msg)
1133         while self.messages:
1134             msg = self.messages.
1135             pop()
1136             conn = self.connectors[:]
1137             while conn:
1138                 out = conn.pop()
1139                 if out.balance < 0:
1140                     out.send(msg)
1141                 else:
1142                     conn.insert(0,out)
1143                     if self.in_channel.
1144                     balance > 0:
1145                         self.messages.
1146                         append(self.in_channel.
1147                         receive())
1148                         stackless.schedule()
1149
1150     def __init__(self, name, in_
1151     channel):
1152         Actor.__init__(self, name,
1153         in_channel)
1154         self.connectors = []
1155         self.messages = []
1156
1157     def action(self, msg):
1158         # dispatch incoming packet
1159         to all connected devices
1160         self.messages.append(msg)
1161         while self.messages:
1162             msg = self.messages.
1163             pop()
1164             conn = self.connectors[:]
1165             while conn:
1166                 out = conn.pop()
1167                 if out.balance < 0:
1168                     out.send(msg)
1169                 else:
1170                     conn.insert(0,out)
1171                     if self.in_channel.
1172                     balance > 0:
1173                         self.messages.
1174                         append(self.in_channel.
1175                         receive())
1176                         stackless.schedule()
1177
1178     def __init__(self, name, in_
1179     channel):
1
```

ets. If the *balance* attribute of a channel is positive, another network node is waiting to send something.

Pickled

squareroot.py in Listing 6 shows a less well known Stackless feature; tasklets can be stored in a binary format – this is referred to as *pickling* in Python speak.

The internal state of the tasklet is kept and can be restored at a later stage. The sample program uses Newton's method to calculate the square root of a number.

squareroot.py generates the output shown below with a

parameter of 2:

```
$ stackless squareroot.py 2
Square root of 2.0
-----
0 : 2.0
1 : 1.5
2 : 1.41666666667
pickle
unpickle
3 : 1.41421568627
4 : 1.41421356237
```

After storing the tasklet, you can run it in another process, on another machine, and possibly even on another architecture. ■

Table 1: Stackless Classes and Functions

<i>task=stackless.tasklet</i> (<function>)	Creates a tasklet <i>task</i>
<i>stackless.schedule</i> ()	Switches to next tasklet
<i>stackless.run</i> ()	Switches to next tasklet and removes itself from the scheduler list
<i>channel=stackless.channel</i> ()	Creates a new channel object <i>channel</i>
<i>channel.send</i> (<i>message</i>)	Sends <i>message</i> to channel; the tasklet blocks until <i>message</i> has been picked up
<i>message=channel.receive</i> ()	Receives <i>message</i> from channel; the tasklet blocks until <i>message</i> has arrived
<i>channel.balance</i>	< 0: Somebody is waiting to receive #new line > 0: somebody is waiting to send

INFO

- [1] Stackless Python: <http://www.stackless.com>
- [2] Erlang: <http://www.erlang.org>
- [3] Oz: <http://www.mozart-oz.org>
- [4] Stack: [http://en.wikipedia.org/wiki/Stack_\(data_structure\)](http://en.wikipedia.org/wiki/Stack_(data_structure))
- [5] Heap: <http://en.wikipedia.org/wiki/Heap>
- [6] Coroutine: <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] Round-robin approach: http://en.wikipedia.org/wiki/Round_robin
- [11] Listings: <http://www.linux-magazine.com/Magazine/Downloads/81>
- [12] Pypy: <http://codespeak.net/pypy/dist/pypy/doc/news.html>

Advertisement