Vorbereitung

- Booten Sie den Rechner unter Linux.
- Melden Sie sich mit Ihrem „`ifw`...“- oder „`ibw`...“-Account an (Passwort ist evtl. die Matrikelnummer).
- Legen Sie (einmalig!) in Ihrem Home-Verzeichnis ein Unterverzeichnis `bspraktikum` an, das geht mit dem Befehl
  `$ mkdir bspraktikum`
- Wechseln Sie in das neue Verzeichnis:
  `$ cd bspraktikum`
- Laden Sie das Archiv mit den Aufgaben-Dateien herunter, das geht mit
  `$ wget http://fhm.hgesser.de/bs-ws2006/prakt1.tgz`
- Entpacken Sie das Archiv mit `tar`:
  `$ tar xzf prakt1.tgz`
- Dadurch entsteht ein neues Unterverzeichnis `prakt1`, in das Sie hinein wechseln:
  `$ cd prakt1`
- Jetzt kann es los gehen...

1. Python kennen lernen

Starten Sie den Python-Interpreter durch Eingabe von **`python`**. Probieren Sie einige der Befehle aus dem Python-Workshop aus:

```
>>> 2+3
>>> range(10)
>>> range(10,20)
>>> range(10) + range(10,20)
>>> for i in range(10): print "Nummer",i
```

(zwei mal [Return] drücken, weil die for-Schleife noch weiter gehen könnte)

Öffnen Sie in einem Browser die Python-Dokumentations-Web-Seite `http://www.python.org/doc/2.4.3/`. Dort finden Sie einen Link „Global Module Index“.

Folgen Sie dem Link und springen Sie dann zum Modul `os`. Abschnitt 6.1.5 beschäftigt sich mit dem Prozess-Management. Lesen Sie die Beschreibungen zu den Funktionen `fork()`, `exec()` und `wait()`. Sie haben diese Funktionen schon in der Vorlesung kennen gelernt, die Referenz soll nur beim Erinnern helfen.

Auch zu allen anderen Standardbibliotheken finden Sie in der Referenz kurze Beschreibungen – das Browser-Fenster sollten Sie beim Programmieren mit Python also immer geöffnet lassen.

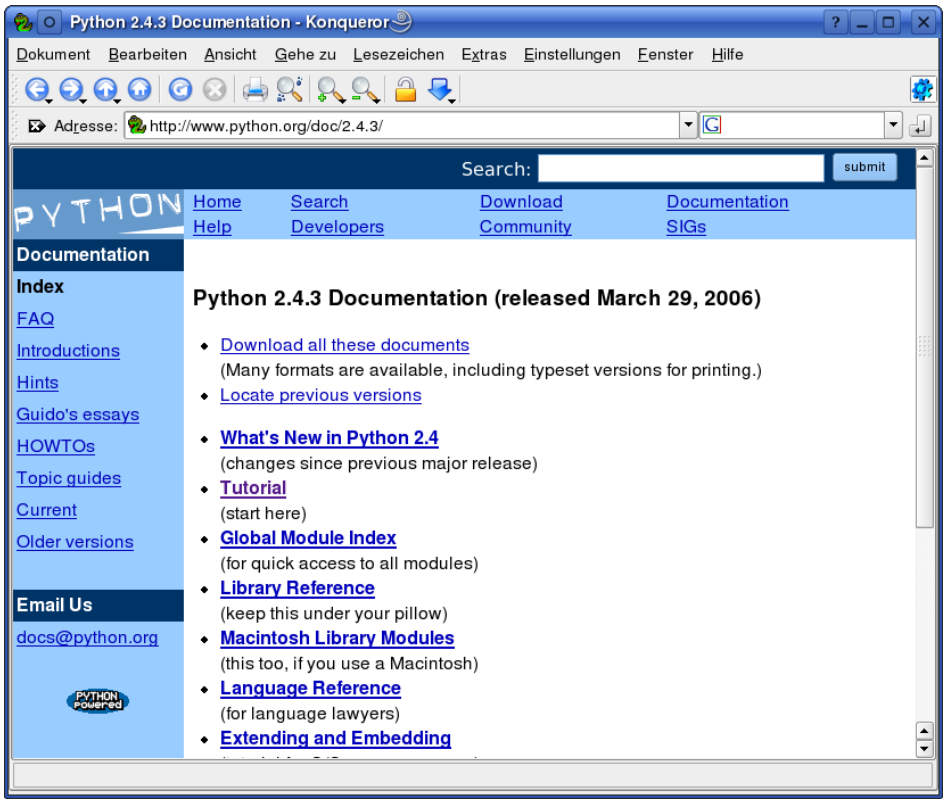

Abbildung 1: Die Dokumentation zu Python 2.4.3

2. Prozess mit fork() erzeugen und externes Programm starten

Vollziehen Sie ein Beispiel aus der Vorlesung nach: Öffnen Sie einen Editor, in dem Sie die neue Datei `hello-world-fork.py` erzeugen.

- Ihr Python-Programm soll zunächst eine „Hello-World"-Botschaft ausgeben und dann mit `fork()` einen Child-Prozess erzeugen;
- der Child-Prozess lädt mit `execl()` das Programm `/bin/ls` und führt es mit dem Parameter `/tmp` aus (um das Inhaltsverzeichnis von `/tmp` auszugeben).
- Währenddessen wartet der Parent-Prozess darauf, dass das Kind seine Arbeit beendet hat. Ist das der Fall, schreibt es noch „Das war's" auf die Konsole.

Das Programm sollte, wenn Sie es mit

```
$ python hello-world-fork.py
```

starten, also eine Ausgabe der Form

```
Hello World
datei1   datei2    datei3    datei4
datei5   datei6    datei7
Das war's
```

erzeugen.

3. Threads mit der `threading`-Bibliothek

In der Vorlesung haben Sie die Bibliothek `thread` kennen gelernt, mit deren Hilfe Sie über die Funktion `start_new_thread()` einfach Threads erzeugen können.

Die `threading`-Bibliothek bietet weit über `thread` hinausgehende Möglichkeiten, mit Threads zu arbeiten.

a) Sie finden im anfangs ausgepackten Verzeichnis `prakt1` eine Python-Datei `watcher.py`. (Die Datei ist auch auf den letzten Seiten ausgedruckt.)
Lesen Sie den Quelltext und versuchen Sie anhand der Dokumentation zum Modul `threading` nachzuvollziehen, was dieses Programm tut. Tipp: Starten Sie das Programm und geben Sie an dem Prompt, den es anzeigt, `help` ein. Experimentieren Sie ein wenig damit herum.

b) Prüfen Sie während der Programmausführung in einem anderen Terminalfenster mit

```
$ ps -eLf
```

wie viele Threads aktiv sind.

c) Erstellen Sie eine Kopie von `watcher.py`:

```
$ cp watcher.py new-watcher.py
```

und nehmen Sie in der neuen Programmdatei einige Änderungen vor:

1. Definieren Sie einen neuen Thread-Typ (genauer: eine abgeleitete Klasse) `outputter`, dessen einzige Aufgabe (in einer Endlosschleife) es ist, sich 3 sek. schlafen zu legen, um dann (abhängig von einer internen booleschen Variablen) Statusinformationen auszugeben oder dies zu lassen. Bei der Initialisierung erzeugen Sie dann mit

```
myoutput = outputter ()
```

einen neuen Thread und starten ihn (`myoutput.start()`).

2. Ergänzen Sie die Schleife im Hauptprogramm so, dass sie einen neuen Befehl „!“ akzeptiert und dann eine noch zu schreibende Funktion `myoutput.toggle()` aufruft.

3. Die Funktion `toggle()` ändert die interne boolesche Variable von `myoutput` (die Sie weiter oben initialisieren müssen) stets von `False` auf `True` oder in die andere Richtung.

4. Beachten Sie, dass beim Programmende auch der Outputter beendet werden muss

Sinn des Ganzen ist, nach dem Definieren mehrerer zu überwachender Dateien auf einen Modus umschalten zu können, bei dem das Programm die Anzeige selbständig aktualisiert.

4. Download-Tool mit Threads

a) Schreiben Sie basierend auf den Bibliotheken `threading` und `httplib` ein Download-Programm, das eine (als Parameter angegebene) Web-Seite mit allen dort eingebundenen Grafiken herunterlädt und alle Dateien in einem Verzeichnis speichert.
Sie können (vereinfachend) davon ausgehen, dass Bilder in den Beispieldateien stets in der Form
`<img src="bild.jpg">`
eingebunden sind – jeweils in einer separaten Zeile ohne weiteren HTML-Code.
Die Beschreibung der nötigen httplib-Funktionen finden Sie in der Python-Dokumentation.

b) Messen Sie die Zeit, die das Programm benötigt, um Daten von einer Beispiel-Web-Seite herunterzuladen – z. B. von http://fhm.hgesser.de/testseite/. Wenn Sie die Messung nicht in Python vornehmen wollen, können Sie das Programm beispielsweise auf der Shell mit dem Tool time starten:

```
$ time python download.py http://fhm.hgesser.de/testseite/
....
real    0m0.014s
user    0m0.002s
sys     0m0.012s
```

c) Das Programm ist nicht besonders effizient, weil es die Bilder sequentiell nacheinander herunterlädt. Starten Sie für jedes Bild einen eigenen Thread und messen Sie erneut die Zeit.

Anhang: Datei `watcher.py`

```
#!/usr/bin/python

# Betriebssysteme, FH Muenchen, H.-G. Esser, WS 2006/07
# watcher.py
# Version 1.0 (2006-11-07)

import os
from threading import Thread
from time import sleep

class watcher(Thread):
   def __init__ (self,filename):
      Thread.__init__(self)
      self.filename = filename
      self.lastline = ""
      self.ExitNow = False
   def exitnow(self):
      self.ExitNow = True
   def run(self):
      while not self.ExitNow:
        try:
           f = os.popen("tail -1 "+self.filename, "r")
           self.lastline = f.readline()
        except:
           self.lastline = "READ ERROR"
        sleep(1)


def add_watcher (filename):
   global watchno, watchers
   if not os.path.exists (filename):
      print "Fehler: Datei existiert nicht"
   else:
      try:
         nul = file(filename).readline()
      except:
         print "Fehler: Datei nicht lesbar"
         return()
      newwatcher = watcher(filename)
      watchers.append(newwatcher)
      watchno+=1
      newwatcher.start()
```

```
def remove_watcher (filename):
   global watchno, watchers
   found = False
   for w in watchers:
      if w.filename == filename:
         w.exitnow()
         watchers.remove(w)
         watchno-=1
         found = True
   if not found:
      print "Fehler: Es gab keinen Watcher fuer diese Datei"

def status ():
   global watchno, watchers
   print "Status: Anzahl Watchers:", watchno
   for w in watchers:
      print w.filename+":", w.lastline[:50]


def printhelp ():
   print  """\
watch <Dateiname>     - Datei in Watch List aufnehmen
unwatch <Dateiname>   - Datei aus Watch List entfernen
status / s            - Statusinformationen ausgeben
help / ?              - diese Hilfe anzeigen
quit                  - Programm beenden"""

watchers = []
watchno = 0

while 1:
   prompt="["+str(watchno)+"]> "
   command = raw_input(prompt)
   if command == "": continue
   try:
      cmd,arg = command.split()
   except ValueError:
      cmd = command
   if   cmd == "watch":        add_watcher (arg)
   elif cmd == "unwatch":      remove_watcher (arg)
   elif cmd in ["status","s"]: status ()
   elif cmd in ["help","?"]:   printhelp ()
   elif cmd == "quit": break
   else: print "Fehler: unbekannter Befehl"


if watchno > 0:
   for w in watchers:
      w.exitnow()
   print "Warte 3 Sekunden..."
   sleep(3)

for w in watchers:
   w.join()
print "Ende"
```