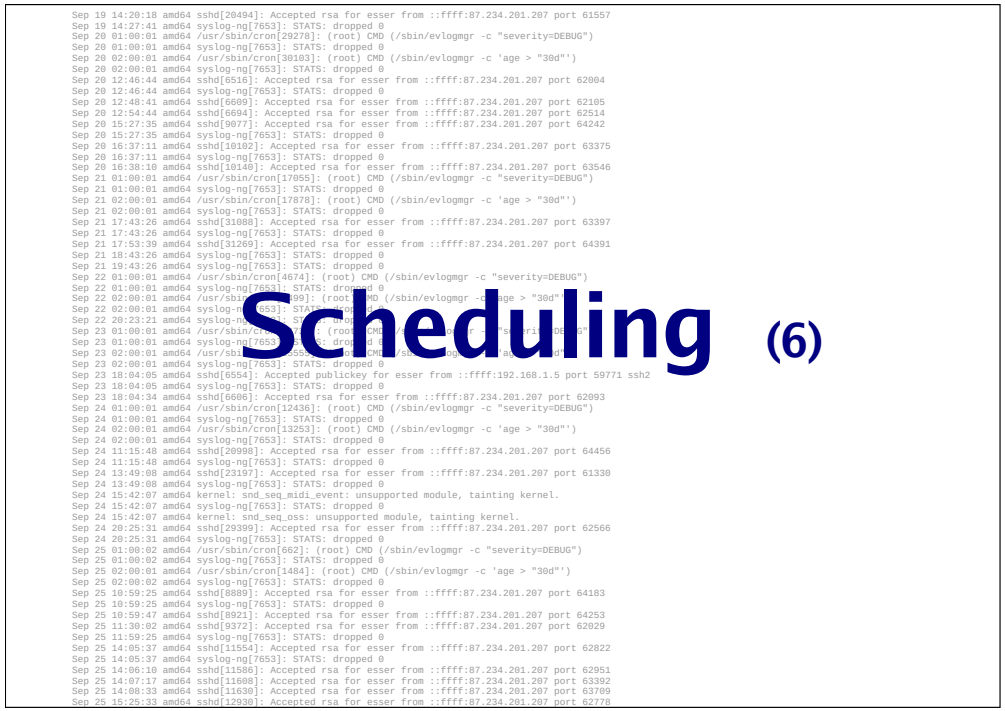


Scheduling (6)

Lösungen Rechner-Praktikum (1)

3. Threads mit der `threading`-Bibliothek

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.

Lösungen Rechner-Praktikum (2)

Lösung Aufgabe 3

```
class outputter(Thread):
   def __init__ (self):
      Thread.__init__(self)
      self.active = False
      self.ExitNow = False

   def toggle(self):
      self.active = not self.active

   def exitnow(self):
      self.ExitNow = True

   def run(self):
      while not self.ExitNow:
        if self.active:
           status ()
        sleep(3)
```


Lösungen Rechner-Praktikum (3)

Lösung Aufgabe 3

Initialisierung (vor der While-Schleife):

```
myoutput = outputter()
myoutput.start()
```

In der Fallunterscheidung:

```
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 == "!":            myoutput.toggle ()
elif cmd == "quit": break
else: print "Fehler: unbekannter Befehl"
```

Beim Aufräumen:

```
myoutput.exitnow()
[...]
myoutput.join()
```

Lösungen Rechner-Praktikum (4)

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. 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.

Lösungen Rechner-Praktikum (5)

Lösung Aufgabe 4 a) ohne Threads

```
#!/bin/env python

import httplib
from threading import Thread
from sys import argv,exit



# Funktion holt ueber eine offene
Verbindung eine Datei
def get_file(conn,path):
  conn.request("GET", path, None)
  result = conn.getresponse()
  return result.read()


# Aufruf-Parameter auswerten
parameter=argv[1:]
if len(parameter)!=1:
  print "Fehler: Falsche Parameterzahl"
  exit(1)
url=parameter[0]
```

```
# URL in Hostname und Pfad aufteilen
split1 = url.split("//")
# evtl. http:// vorhanden?
if len(split1)>1:
  url=split1[1]  # jetzt nur noch
www.xyz.de/path

try:
  [server,path] = url.split("/",1)
  path="/"+path
except:
  server=url    # Hier der Fall, dass
                # gar kein Pfad folgt
  path="/"
print "Server: '"+server+"'"
print "Pfad:   '"+path+"'"
```


Lösungen Rechner-Praktikum (6)

Lösung Aufgabe 4 a) ohne Threads

```
# Web-Seite laden
connection =
httplib.HTTPConnection(server)
pagetext=get_file(connection,"/"+path)

if path[-1]=="/":
  localfilename="index.html"
else:
  localfilename=path.split("/")[-1]
                # letzter Teil des Pfads
htmlfile=file(localfilename,"w")
htmlfile.write(pagetext)
htmlfile.close()

lines=pagetext.split("\n")
```

```
imagenames=[]
for l in lines:
  if l.startswith("<img"):
    print l
    (x,name,y)=l.split('"')
    if not name.startswith("/"):
      name=path+name
    print name
    imagenames.append(name)

for image in imagenames:
  localfilename=image.split("/")[-1]
            # letzter Teil des Pfads
  print "lokaler Dateiname:",
                      localfilename
  content=get_file(connection,image)
  imagefile=file(localfilename,"w")
  imagefile.write(content)
  imagefile.close()

connection.close()
```


Lösungen Rechner-Praktikum (7)

Lösung Aufgabe 4 c) mit Threads

```
#!/bin/env python
from threading import Thread
[...]
# Thread holt vom Server eine Datei und
# speichert sie lokal
# (nicht moeglich, die Funktion get_file als
# Klasse zu verwenden: HTTPConnection kommt
# nicht mit mehreren parallelen Requests aus
# mehreren Threads klar)
class get_file_thread(Thread):
   def __init__ (self,server,path, localname):
      Thread.__init__(self)
      self.server=server
      self.path=path
      self.localname=localname
   def run(self):
      conn = httplib.HTTPConnection(self.server)
      conn.request("GET", self.path, None)
      result = conn.getresponse()
      content=result.read()
      conn.close()
      imagefile=file(self.localname,"w")
      imagefile.write(content)
      imagefile.close()
```

```
[...]


downloaders=[]

for image in imagenames:
  localfilename=image.split("/")[-1]
          # letzter Teil des Pfads
  print "lokaler Dateiname:",
                  localfilename
  newthread=get_file_thread(server,
              image,localfilename)
  newthread.start()
  downloaders.append(newthread)


for d in downloaders:
  d.join()
```

Lösungen Rechner-Praktikum (8)

Lösung Aufgabe 4
Performance-Vergleich (Seite mit 6 Bildern)

```
$ time download-no-threads.py ...
[...]
real    0m0.804s
user    0m0.088s
sys     0m0.036s
```

Durchschnitt: 0,7 s

9 Bilder: 1,2 s

30 Bilder: 2,9 s

```
$ time download.py ...
[...]
real    0m0.370s
user    0m0.096s
sys     0m0.040s
```

Durchschnitt: 0,4 s, **Speedup 75 %**

9 Bilder: 0,7 s, **Speedup 71 %**

30 Bilder: 1,4 s, **Speedup 107 %**

Linux O(1) Scheduler (Fortsetzung)

Linux O(1) Scheduler (5)

- Für jede CPU eine separate Warteschlange
- 140 Prioritätslevel,
  kleiner Wert = hohe Priorität:
  - 1-100: Realtime-Prozesse (MAX_RT_PRIO=100)
  - 101-140: Normale Prozesse (MAX_PRIO=140)
- Normale Tasks
  - haben Nice-Wert n ($-19 \leq n \leq 20$),
  - Prio = MAX_RT_PRIO + n + 20,
  - erhalten Zeitquantum

Linux O(1) Scheduler (6)

- Echzeit-Tasks
  - statische Priorität
  - zwei Klassen:
    - FIFO (ohne Unterbrechungen) und
    - Round Robin (mit Zeitquanten)
- Interaktivitätsschätzer:
  prüft, ob ein Prozess interaktiv ist – wenn ja,
  erhält er eine höhere Priorität
  (nur für normale Prozesse, nicht Echtzeit)

Linux O(1) Scheduler (7)

Für jede CPU und jede Priorität eine Warteschlange (also 140 Listen pro CPU)!

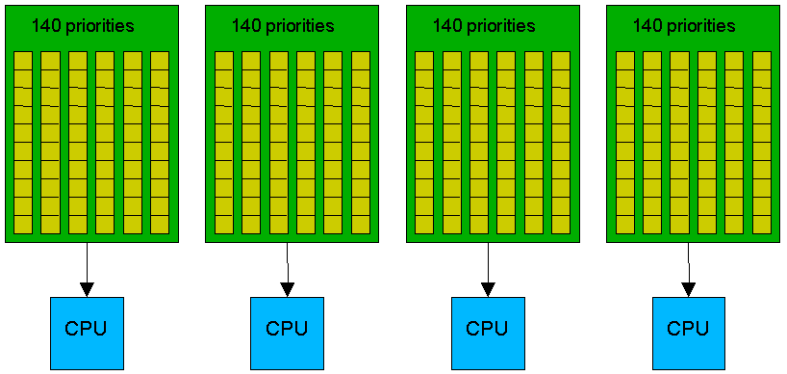


Bild: Linux Journal, http://www.linuxjournal.com/node/7178/

Linux O(1) Scheduler (8)

Nächsten Prozess finden ist sehr einfach:

- Jede CPU muss nur in ihrer privaten Prozessliste suchen
- Bitmap speichert, welche (der 140) Queues leer sind – Suche der Form „1. Bitmap-Feld mit Wert 1“ geht schnell
- Innerhalb der so gefundenen Liste einfach den ersten Prozess wählen
- Suchoperation hängt zwar „von 140“ ab, aber nicht von der Anzahl der Prozesse -> O(1)

Linux O(1) Scheduler (9)

Zusätzlich zu Runqueue gibt es eine „Expired Runqueue“

- aktiver Prozess, dessen Quantum ausläuft, wird unterbrochen und in die Expired Queue verschoben
- beim Verschieben berechnet der Scheduler Quantum und Priorität für diesen Prozess neu (sortiert ihn also evtl. auf eine andere Prioritätsstufe ein).
- Ist die Runqueue komplett leer, werden Runqueue und Expired Runqueue vertauscht

Linux O(1) Scheduler (10)

Interaktivitätsschätzer

- Scheduler versucht zu erkennen, ob Prozesse I/O- oder CPU-lastig sind
- Metrik: Verhältnis Rechenzeit zu (I/O-) Wartezeit
- Scheduler
  - belohnt I/O-lastige Prozesse
  - bestraft CPU-lastige Prozesse

bis zu +/- 5 Punkte bei Prior.-Berechnung

Linux O(1) Scheduler (11)

Load Balancer

- Eigentlich: CPU-Wechsel vermeiden, da CPU-Cache unbrauchbar wird
- Andererseits: CPUs, die längere Zeit idle sind, sind noch schlimmer
- Alle 200 ms prüft eine CPU, ob die Lastverteilung ungleichmäßig ist; wenn ja, werden die Prozesse neu verteilt
- Problem: Behandlung von HyperThreading-CPUs mit virtuellen CPUs

Performance Linux 2.4 / 2.6

Hackbench: bis zu 200 Client/Server-Prozesse

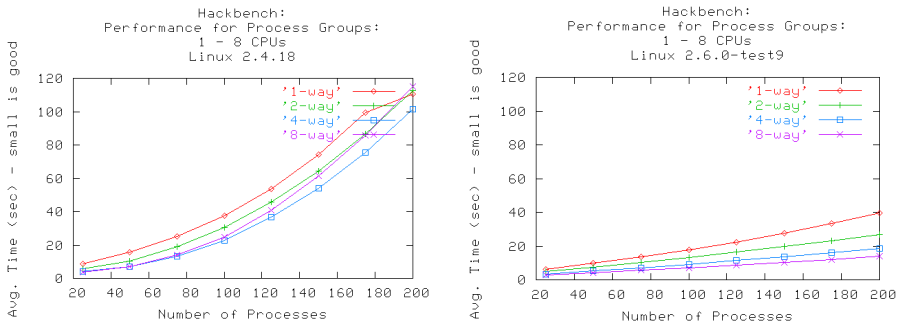


Bild: http://developer.osdl.org/craiger/hackbench/

„Desktop Scheduling“

Scheduler für Multimedia

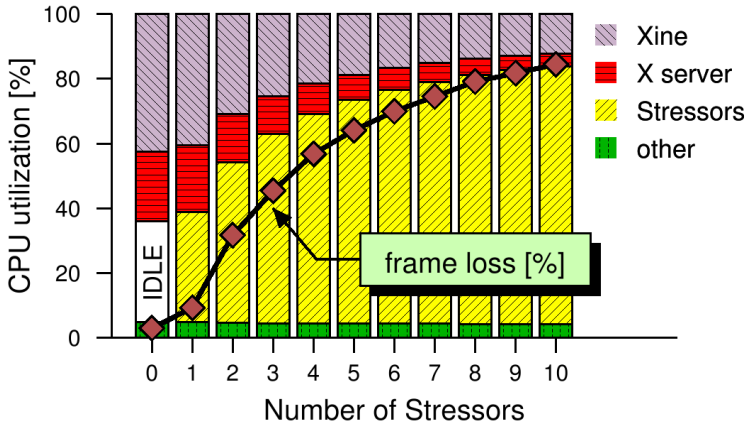
- Die meisten Scheduler unterscheiden klassisch:
  - I/O-lastig (interaktiv) vs.
  - CPU-lastig (nicht-interaktiv)
- niedrige CPU-Nutzung -> hohe Priorität
- Problem: Multimedia-Anwendungen (Video, Spiele) brauchen viel CPU-Zeit, sind also nicht von klassischen Hintergrundjobs zu unterscheiden -> sie laufen so nicht gut

„Desktop Scheduling“

Performance-Test:

- „Stressor-Programme“: stark CPU-lastig, z.B. Kernel kompilieren
- Wie verhalten sich Multimedia-Programme, wenn die Anzahl der Stressor-Programme wächst?

„Desktop Scheduling“



Klassisches Scheduling (Bild: [1])

[1] Etsion, Tsafrir, Feitelson, „Desktop Scheduling: How can we know what the user wants?“, 14th ACM Intl. Workshop on Network & Operating Systems Support for Digital Audio & Video (NOSSDAV), pp. 110-115, Jun 2004, www.cse.huji.ac.il/~dants/papers/HuCpri04NOSSDAV.pdf

„Desktop Scheduling“

Alte Ansätze zum Erkennen von Interaktivität:

- Programme sagen selbst über sich aus, ob sie interaktiv sind
- Anwender entscheidet (über Startparameter oder über einen Prozessmonitor), welche Programme mit hoher Priorität laufen sollen
- Programmfenster, das den Fokus hat, erhält mehr Rechenzeit (so macht es Windows)

„Desktop Scheduling“

Neuer Ansatz zum Erkennen von Interaktivität:

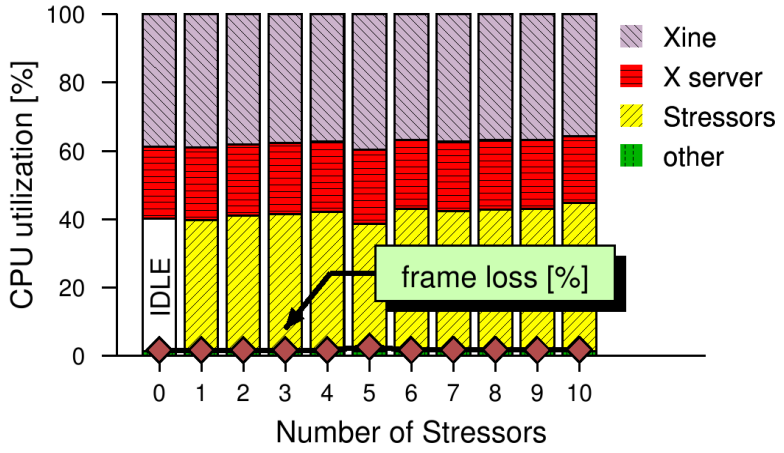
- Einige I/O-Geräte werden besonders häufig von interaktiven Programmen genutzt, z.B.
  - Tastatur: offensichtlich interaktiv
  - Videokarte: Wenn ein großer Bereich des Bildschirms permanent aktualisiert wird, ist das wohl auch interaktiv
- Über Statistiken der I/O-Geräte herausfinden, welche Prozesse für den Anwender „wichtig“ sind

„Desktop Scheduling“

HuC (human centered) Devices

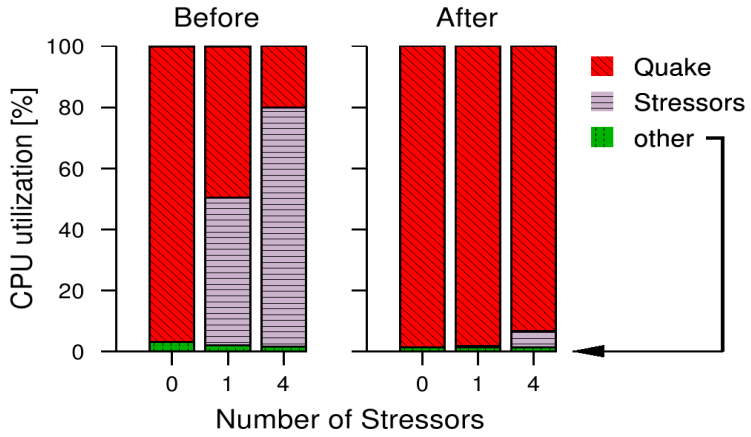
- Nur bestimmte I/O-Geräte interessant, z. B. Tastatur, Maus, Bildschirm, Joystick, Soundkarte -> **HuC Devices**
- Für den Bildschirm: X-Server patchen
  - Client-spezifische I/O-Traffic-Daten sammeln
  - 1x/sec an den Scheduler weiterleiten
  - Maßstab: Wie viel % der Bildschirmanzeige haben sich geändert?

„Desktop Scheduling“



HuC-Scheduler liefert bessere Ergebnisse Bild: [1]

„Desktop Scheduling“



Wie viel Rechenzeit erhält Quake? Bild: [1]
links: normaler Linux-Scheduler, rechts: HuC-Scheduler