Vorbereitung

- Booten Sie den Rechner unter Linux, melden Sie sich an und öffnen Sie ein Terminalfenster (gnome-terminal, xterm etc.).
- Die Dateien zum heutigen Praktikumstermin laden Sie von der Vorlesungsseite herunter: `wget http://fhm.hgesser.de/bs-ss2008/prakt05.tgz`
- Entpacken Sie das Archiv (`tar xzf prakt05.tgz`) und wechseln Sie mit `cd prakt05` in das neue Unterverzeichnis `prakt05`.

1. Verschiedene Scheduler in Python simulieren

Im Archiv finden Sie das Python-Programm `sched.py`, das in der Funktion `schedule()` den in der Vorlesung vorgestellten FCFS-Scheduler (First Come, First Served) implementiert:

```
def schedule():
  # Implementation des FCFS-Schedulers (First Come, First Served)
  global current, tasks, runqueue, blocked, current, cputime

  # falls aktueller Prozess noch bereit: weitermachen (FCFS)
  if (current!=-1) and (get_status(current) == S_ACTIVE):
    return current

  # falls weder bereite noch blockierte Prozesse: Ende
  if (runqueue == []) and (blocked == []):
    # vielleicht kommt noch einer?
    if futureprocesses(cputime) == []: return -2

  # falls nicht: nehme ersten Prozess aus Runqueue
  if (runqueue != []):
    return runqueue[0]
  else:
    # falls alle blockiert: idlen!
    return -1  #  alle blockiert
```

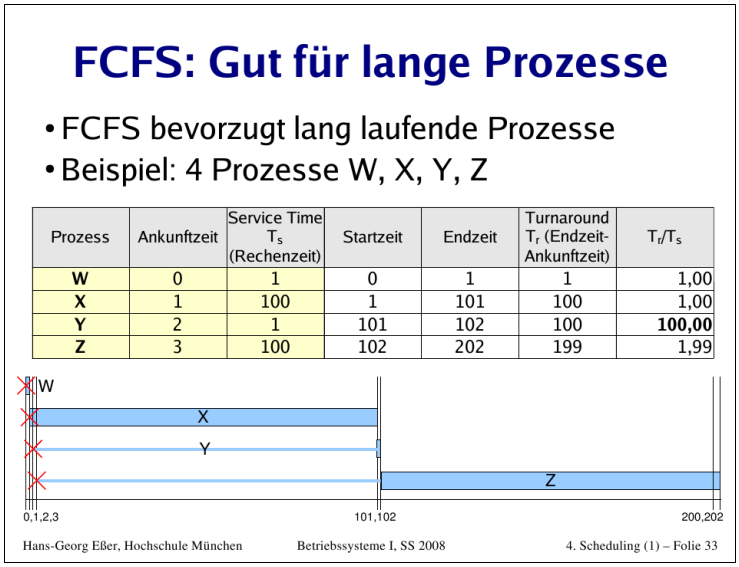
Sie rufen es mit dem Befehl `python sched.py` *`Datensatz.dat`* auf, wobei Sie *`Datensatz.dat`* durch den Namen einer Konfigurationsdatei ersetzen, welche die Prozesskonfiguration enthält. Diese Dateien haben folgenden Aufbau:

```
1:1,6,3,4,-1
2:30,-1
4:4,1,4,1,4,-1
```

Dabei steht jede Zeile für einen Prozess. Am Anfang einer Zeile findet sich der Startzeitpunkt des Prozesses, dann folgen nach einem Doppelpunkt (durch Kommata getrennt) die Zeiten, in denen der Prozess CPU- und I/O-Phasen durchläuft – die erste Phase ist dabei eine CPU-Phase.

Ein Prozess, der mit I/O beginnt, müsste also in der Zeitenliste zunächst den Wert `0` (für eine CPU-Phase der Länge 0) enthalten. Außer direkt am Anfang darf in der Zeitenfolge keine `0` vorkommen – sie wäre redundant: Die Abfolge ...`,3,0,4,`... müsste zu ...`,7,`... zusammengefasst werden. Der letzte Eintrag (`-1`) in jeder Zeile kennzeichnet das Prozessende.

a) Überprüfen Sie anhand der Konfigurationsdatei `nur-cpu.dat` die Ausführreihenfolge für die vier Beispielprozesse auf Folie 33 der ersten Scheduling-Vorlesung (`bs-ss2008-esser-05.pdf`, siehe Abbildung rechts).



FCFS: Gut für lange Prozesse

- FCFS bevorzugt lang laufende Prozesse
- Beispiel: 4 Prozesse W, X, Y, Z

Prozess	Ankunftzeit	Service Time T_s (Rechenzeit)	Startzeit	Endzeit	Turnaround T_r (Endzeit-Ankunftzeit)	T_r/T_s
W	0	1	0	1	1	1,00
X	1	100	1	101	100	1,00
Y	2	1	101	102	100	**100,00**
Z	3	100	102	202	199	1,99

Hans-Georg Eßer, Hochschule München Betriebssysteme I, SS 2008 4. Scheduling (1) – Folie 33

b) Machen Sie sich mit dem Programmquelltext vertraut – er ist recht ausführlich dokumentiert.
Welche Aufgaben übernehmen die Funktionen `create_process`, `run_current` und `update_blocked_processes`?

c) Erzeugen Sie eine Kopie von `sched.py` (etwa: `sched-sjf.py`) und passen Sie darin die Funktion `schedule()` so an, dass sie statt FCFS den SJF-Scheduler (Shortest Job First) implementiert.

d) Etwas komplizierter wird es, einen unterbrechenden Scheduler zu schreiben. Im letzten Schritt passen Sie eine Kopie von `sched-sjf.py`, z. B. `sched-srt.py`, an und implementieren nur den SRT-Scheduler (Shortest Remaining Time). Hier funktioniert die einfache Variante aus den vorigen zwei Schedulern nicht mehr, aktive Prozesse solange laufen zu lassen, bis sie sich beenden oder in eine I/O-Phase eintreten.

e) Wenn noch Zeit übrig ist, können Sie sich bereits Gedanken über die Implementation der Scheduler für interaktive Betriebssysteme machen (Round Robin, Virtual RR, Prioritäten- und Lotterie-Scheduler) – um diese wird es am nächsten Praktikumstermin gehen.

[☛ Abgabe per Mail] Die beiden von Ihnen erstellten Programmdateien `sched-sjf.py` und `sched-srt.py` schicken Sie bitte per E-Mail an `hans-georg.esser@hm.edu`. Bitte schreiben Sie in diese Mail auch die Antwort zu Aufgabe 1 b). Vergessen Sie nicht, den von Ihnen erstellten Quellcode gut zu dokumentieren.

Wenn Sie beim Testen feststellen, dass sich Ihr Programm nicht wie gewünscht verhält, Sie dies aber nicht beheben können, geben Sie in der Mail an, inwiefern Ihr Programm sich falsch verhält und was Sie als Ursache vermuten.