



Vorbereitung

- Die Dateien zum heutigen Praktikumstermin laden Sie von der Vorlesungsseite herunter:
wget <http://hm.hgesser.de/bs-ss2011/prakt07.tgz>
- Entpacken Sie das Archiv und wechseln Sie in das neue Unterverzeichnis `prakt07`.

18. 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
```

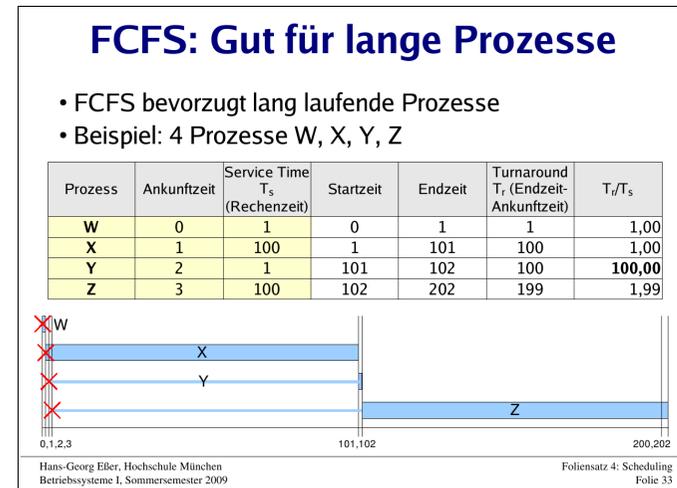
Dabei steht jede Zeile für einen Prozess. Am Anfang einer Zeile findet sich der Startzeitpunkt des Prozesses, dann folgen nach einem Doppelpunkt (durch Kommata getrennt) die Zeiten, in denen der Prozess CPU- und I/O-Phasen durchläuft – die erste Phase ist dabei eine CPU-Phase.

Ein Prozess, der mit I/O beginnt¹, 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.

¹ Ob es sinnvoll ist, Prozesse mit einer I/O-Phase beginnen zu lassen, ist dabei noch eine interessante Frage – ich würde sagen: Nein, ist es nicht.



- a) Überprüfen Sie anhand der Konfigurationsdatei `nur-cpu.dat` die Ausführreihenfolge für die vier Beispielprozesse auf Folie 33 von Foliensatz 4 (`bs-ss2011-esser-04-4up.pdf`, siehe Abbildung).



- b) Machen Sie sich mit dem Programm Quelltext 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 Schemen 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 Round Robin, Prioritäten- und Lotterrie-Scheduler) – um diese wird es am nächsten Praktikumstermin gehen.

Dokumentieren Sie den von Ihnen erstellten Quellcode.

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.

Die Lösungen schicken Sie mir bitte per E-Mail bis zum 19.05.2011 mit dem Betreff „BS-2011, Übung 7, NAMEN“.