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/prakt06.tgz`
- Entpacken Sie das Archiv (`tar xzf prakt06.tgz`) und wechseln Sie mit `cd prakt06` in das neue Unterverzeichnis `prakt06`.

1. Round-Robin- und VRR-Scheduler

Aus dem Quellcode-Archiv entnehmen Sie die Datei `sched.py`, die einen FCFS-Scheduler implementiert. (Es handelt sich um eine leicht angepasste Variante des Scheduler-Programms aus dem letzten Praktikum. Diese verwendet jetzt die Funktion `activate`, um einen Prozess aktiv zu machen.)

a) Verwenden Sie diese Datei als Ausgangsbasis für einen Round-Robin- (RR-) Scheduler – FCFS und RR unterscheiden sich nur dadurch, dass RR nach Ablauf eines vorgegebenen Zeitquantums den laufenden Prozess unterbricht und ans Ende der Warteschlange anhängt.

   Beachten Sie für Ihre Lösung die folgenden Punkte:

   - Die Länge des Quantums sollte sich über eine Variable einstellen lassen; alternativ lesen Sie über `argv[2]` ein zweites Argument aus, das der Scheduler als Länge des Quantums interpretiert.
   - Damit Prozesse nie mehr Zeit als das Quantum verbrauchen, erweitern Sie den Prozesskontrollblock (also jeweils das Dictionary `tasks[i]`) um einen Zähler. Prüfen Sie, wann genau Sie diesen Zähler zurück auf 0 setzen müssen – neben den offensichtlichen Fällen (Initialisierung und Entzug der CPU durch den Scheduler) gibt es noch eine andere Situation. Der zur Verfügung gestellte Quellcode enthält bereits Funktionen zum Auslesen, Schreiben und Erhöhen dieses Zählers `usedquant`.

b) In der Vorlesung haben Sie gesehen, dass der Standard-RR-Scheduler CPU-lastige Prozesse bevorzugt. Die Variante Virtual Round Robin (VRR) behebt diese ungleiche Behandlung von I/O- und CPU-lastigen Prozessen. Erzeugen Sie eine Kopie Ihres RR-Schedulers, z. B. `sched-vrr.py`, und ergänzen Sie die bevorzugte Warteschlange, in die Prozesse wechseln, die eine I/O-Operation starten, ohne ihr volles Quantum aufgebraucht zu haben.

c) Testen Sie beide Scheduler mit dem Beispiel aus der Vorlesung, das wir dort zur Unterscheidung von RR und Virtual RR verwendet haben prüfen Sie, ob Ihr VRR-Scheduler wirklich I/O-lastige Prozesse besser stellt als der RR-Scheduler.

d) Um die für Context Switches benötigte Rechenzeit mit in die Untersuchungen einzubeziehen, erweitern Sie den RR-Scheduler aus Aufgabenteil a) um einen Context-Switch-Zähler. Jedesmal, wenn der Scheduler die Entscheidung trifft, einen anderen Prozess auszuwählen, oder wenn wegen des Wechsels in einer I/O-Phase ein anderer Prozess an die Reihe kommt, soll dieser Zähler erhöht werden. Außerdem soll dann auch die Uhr hochgezählt werden (nehmen Sie an, dass ein Context Switch eine Zeiteinheit benötigt), so dass die Context-Switch-Zeiten auch in den Gesamtrechenzeiten der Prozesse auftauchen.

2. Lotterie-Scheduler

Beim Lotterie-Scheduling hat jeder Prozess eine spezifische Anzahl an Losen. Diese Information müssen Sie in die Konfigurationsdatei übernehmen und im Scheduler bei der Initialisierung auslesen. Ein mögliches Format für die Zeilen in der Konfigurationsdatei wäre

`s,l:c,i,c,i,...,-1` (`s`=Startzeit, `l`=Anzahl Lose, `c`/`i`: Längen der CPU- und I/O-Phasen)

anstelle von

`s:c,i,c,i,...,-1.`

Die Gesamtzahl aller Lose ergibt sich dann aus der Summe aller l-Werte.

Verwenden Sie wieder den FCFS-Scheduler als Ausgangsbasis, um nun das in der Vorlesung vorgestellte Lotterie-Scheduling-Verfahren umzusetzen. Python bietet das Modul `random`, das verschiedene Funktionen enthält, mit denen Sie zufällige Werte abrufen können. Experimentieren Sie mit `randrange()` der Aufruf`randrange(n)` gibt einen Wert zwischen `0` und `n-1` zurück, mit anderen Worten: ein zufälliges Element aus `range(0,n) = [0,1,...,n-1]`.

Auch der Lotterie-Scheduler lässt einen ausgewählten Prozess mehrere Zeiteinheiten (ein Quantum) lang laufen hier können Sie die Vorarbeiten für den RR-Scheduler übernehmen, insbesondere das Speichern der bereits genutzten Zeit eines Prozesses.

3. Prioritäten-Scheduler

Erweitern Sie die Prozessstruktur um statische Prioritätswerte zwischen 1 und 5, wobei ein höherer Wert höhere Priorität (Prozess ist wichtiger) bedeutet. Dazu müssen Sie den Parser anpassen, der nun Zeilen im Format

`s,p:c,i,c,i,...,-1` (`s`=Startzeit, `p`=Priorität, `c`/`i`: Längen der CPU- und I/O-Phasen)

anstelle von

`s:c,i,c,i,...,-1`

einlesen soll. Erzeugen Sie zwei klassische Implementationen eines Schedulers, der diese Prioritäten beachtet:

a) Der Scheduler führt grundsätzlich einen Prozess mit höchster Priorität aus.

b) Der Prozessor erhöht bei Prozessen mit hoher Priorität die Laufzeit (das Quantum), und zwar durch Multiplikation mit der Priorität. Ein Prozess der Priorität 2 erhält also jeweils zwei volle Quanten.

4. Prioritäten vs. Lotterie

Können Sie (statische) Prioritäten und Losvergabe durcheinander simulieren? Beschreiben Sie kurz für beide Richtungen, wie Sie dazu vorgehen würden.

[☛ Abgabe per Mail] Die beiden von Ihnen erstellten Programmdateien schicken Sie bitte per E-Mail an `hans-georg.esser@hm.edu`. 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.