KV Betriebssysteme	Übung #4	SS 2005
Name:	Matr-Nr:	Gruppe:
Abgabe: 24.05.2005		Tutor:

Die Abgabe der gesamten Ausarbeitung via WeLearn ist unbedingt erforderlich!

Beispiel 1: Simulation von CPU Scheduler Algorithmen

Für diese Übung bekommen Sie ein Framework zur Simulation von CPU Scheduler Algorithmen für zwei parallel arbeitende CPUs bereits „fertig" zur Verfügung gestellt. Dieses stellt die Basis für Ihre Übungsaufgaben dar (Framework-Download von der LVA Homepage).

Der Simulationsverlauf selbst ist einfach gehalten:

- → Nur Prozesse (keine Threads) werden simuliert
- → Die einzigen Ressourcen, auf welche Prozesse eventuell warten müssen, sind die beiden CPUs.

Allgemeines zu den Übungsaufgaben und dem Framework:

- → Obwohl Sie den Source Code des Frameworks zur Verfügung gestellt bekommen, sehen Sie dieses als „unveränderlich" an (denken Sie z. B. an kommerzielle Frameworks, die nur in compilierter Form zur Verfügung gestellt werden). D. h., für Ihre Erweiterungen müssen die angebotenen, dokumentierten Schnittstellen ausreichen.
- → Verwenden Sie für neu erstellte Klassen das Package `at.jku.fim.cpuscheduling.student`

Aufgabe 1 (2+2 Punkte)

In der Klasse `at.jku.fim.cpuscheduling.student.VirtualPCHelper` sind zwei Methoden zu implementieren, die das dynamische Instanziieren von Strategie-Klassen übernehmen:

a) Prozess-Erzeugung

Für Simulationen sind häufig „zufällige" Ereignisse, Interaktionen, etc. wichtig. Bei unserer CPU Scheduler Simulation müssen u. a. zufällig Prozesse erzeugt werden, welche vom Scheduler bearbeitet werden. Das Erzeugen von Prozessen ist als „Strategie-Muster" implementiert. D. h., die Strategie (der Algorithmus) für das Erzeugen von Prozessen ist austauschbar. Alle Strategien haben die Gemeinsamkeit, dass Sie von der abstrakten Basis-Klasse

`at.jku.fim.cpuscheduling.processcreation.ProcessCreator`

abgeleitet sind. Der Benutzer kann die gewünschte Strategie-Klasse zur Laufzeit mittels eindeutigem Klassennamen (Packagename + Klassenname) spezifizieren. Ihre Aufgabe ist es nun, von einer Klasse dynamisch eine Instanz zu erzeugen und diese als Methodenergebnis bereitzustellen. Testen Sie zusätzlich ab, ob die spezifizierte Klasse auch tatsächlich von der

genannten Basisklasse ProcessCreator abgeleitet ist. Sollten Fehler auftreten, melden Sie diese mittels *IllegalArgumentException* dem umgebenden Framework.

Testen Sie Ihre Implementierung mit folgenden beiden Klassen :

at.jku.fim.cpuscheduling.processcreation.ProcessCreatorImpl

at.jku.fim.cpuscheduling.processcreation.SimpleProcessCreator

(ProcessCreatorImpl ist die stdm. geladene Strategie, die sich vielfältig konfigurieren lässt. SimpleProcessCreator erzeugt alle X Einheiten einen Prozess, wobei X konfigurierbar ist).

Zu implementierende Methode in *VirtualPCHelper*:

```
public static ProcessCreator getProcessCreatorInstance
```

b) Strategie

Dasselbe Vorgehen wie bei a) ist notwendig für das Laden von Scheduler Klassen. Jede Scheduler-Implementierung muss von der Klasse

at.jku.fim.cpuscheduling.scheduling.Scheduler

abgeleitet sein. Gehen Sie bei der Implementierung analog wie bei a) vor.

Zu implementierende Methode:

```
public static Scheduler getSchedulerInstance
```

Aufgabe 2 (5 Punkte)

Informieren Sie sich aus der JavaDoc über die abstrakte Klasse Scheduler und ermitteln Sie, welche Methoden mindestens für eine Scheduler-Implementierung vorhanden sein müssen.

at.jku.fim.cpuscheduling.scheduling.FifoBatchScheduler ist eine sehr einfache Beispielimplementierung. Wird dieser Scheduler eingesetzt, beschränkt sich das Multiprocessing auf zwei Prozesse (jeweils einer auf einer CPU) gleichzeitig. Die Prozesse werden im Batch-Verfahren (nicht unterbrechend) abgearbeitet. D. h., erst wenn ein Prozess fertig abgearbeitet ist, bekommt der nächste die CPU. Für heutige Betriebssysteme, wo viele Prozesse gleichzeitig laufen, die allesamt die CPU meist nicht voll beanspruchen, ist diese Lösung natürlich nicht praktikabel.

Implementieren Sie einen einfachen RoundRobin Scheduler. Die Länge der Zeitscheibe soll hierbei vom Benutzer als Parameter konfigurierbar sein (Integer-Werte von 3 bis 300 sollen akzeptiert werden). Die Prioritäten der Prozesse sollen noch unberücksichtigt bleiben. Beachten Sie, dass Prozesse während Ihres Lebenszyklus möglichst wenige CPU-Wechsel durchmachen sollen (Caching!). Dokumentieren Sie weiters Ihre Lösungsansätze textuell in einem separaten Beiblatt (Ausarbeitung soll mind. ½ Seite umfassen).

Aufgabe 3 (9 Punkte)

Implementieren Sie einen Scheduler, welcher auch Prozess-Prioritäten berücksichtigt (basierend auf RoundRobin oder verwenden Sie Ihr eigenes Verfahren). Bedenken Sie, dass Ihr Algorithmus einerseits für eine optimale Nutzung der Ressourcen sorgen soll, andererseits aber auch möglichst effizient und schnell arbeiten muss. Der Benutzer soll zur Laufzeit die Möglichkeit haben, den Scheduler dynamisch zu beeinflussen. Überlegen Sie, welche Parameter in Ihrem Algorithmus für den Simulationsbenutzer interessant sind, dynamisch verändern zu können.

Beachten Sie bitte folgende Punkte:

- → Dokumentieren Sie Ihre Überlegungen und Lösungen textuell in einem separaten Beiblatt (Ausarbeitung soll mindestens eine Seite umfassen).
- → Überlegen Sie, ob und wann CPU Wechsel von Prozessen vertretbar sind (Caching-Vorteil geht verloren)
- → Begründen Sie, ob Sie in Ihrer Implementierung „Starvation“ extra berücksichtigen müssen oder nicht.
- → Wie gehen Sie bei nicht-gesetzter Priorität vor?
- → Lassen Sie den Benutzer mit Parametern eingreifen, d. h., machen Sie Ihren Scheduler möglichst flexibel und parametrisierbar. Dokumentieren Sie die Parameter.
- → Schätzen Sie die Laufzeitkomplexität Ihres Schedulers ab (wieviele Iterationen über die READY-Queue - der Scheduler soll nicht mehr CPU Zeit benötigen wie die eigentlichen Prozesse ☺)

Beispiel für mögliche Optimierungen:

→ Überlegen Sie, ob vielleicht mehrere READY-Queues (z. B. eine für jede Prioritätskategorie) sinnvoll sind.

→ Beachten Sie auch die Möglichkeit, dass laufende Prozesse der CPU mit der Methode `interrupt` vorzeitig entzogen werden können (z. B. bei Eintreffen eines hochprioritären Prozesses). Danach wird automatisch wieder der Scheduler für die CPU aufgerufen.

Testen Sie Ihren Scheduler mit verschiedenen Konfigurationen (verschiedene Scheduler Konfigurationen, verschiedenen Process Creator Konfigurationen) und dokumentieren Sie die Ergebnisse.

Aufgabe 4 (6 Punkte)

Zur Auswertung der Ergebnisse von verschiedenen Scheduler-Implementierungen ist ein Statistik-Modul vorgesehen, welches in dieser Aufgabe zu implementieren ist. Auf Knopfdruck sollen die Daten für die Statistik ausgewertet und in einer JTable angezeigt werden. Im User-Interface sind bereits die dafür notwendigen Komponenten vorgesehen.

Vervollständigen Sie die Klasse

`at.jku.fim.cpuscheduling.student.StatisticsTableModel`.

Beim Aufruf von `refreshData()` soll die aktuelle Statistik berechnet und in einem statischen Array für späteres Anzeigen gespeichert werden. In die Statistik sollen nur bereits abgeschlossene Prozesse miteinberechnet (`VirtuallPC.getFinishedList()`) werden. Die Daten müssen nach dem Muster - wie in `AbstractTableModel` (JavaDoc!) vorgesehen - bereitgestellt werden.

Überlegen Sie, wie sie die Werte übersichtlich in tabellarischer Form darstellen können!

Folgende Werte sollen jeweils aggregiert für die 4 Prioritätsstufen und einmal für alle Prozesse aggregiert berechnet werden:

- → Anzahl der Prozesse
- → Wieviel Prozent der Gesamtlebenszeit der Prozesse (von Erstellen des Prozesses bis zur Fertigstellung) wurde in der Warteschlange verbracht?
- → Wieviele Unterbrechungen haben die Prozesse durchschnittlich durchlebt (Ready-Waiting)?
- → Wieviele CPU-Wechsel haben die Prozesse durchschnittlich durchlebt?
- → Wie lange war die durchschnittliche Zeitspanne bis zur ersten CPU-Zuteilung?

Zusätzlich sollen noch folgende Werte berechnet werden:

- → Auslastungen der CPUs
- → Durchschnittliche Anzahl von Prozessen in der READY-Queue (Die Verwaltung der READY-Queue ist Aufgabe von Scheduler-Implementierungen. D. h., die Funktionalität muss von Ihrer Scheduler-Implementierung zur Verfügung gestellt werden. Die Kennzahl der Auslastung kann also nur bei bestimmten Scheduler Klassen angezeigt werden.)

Überlegen Sie, welche Kennzahlen für Ihren Scheduler aus Aufgabe 3 noch interessant wären und dokumentieren Sie das Ergebnis Ihrer Überlegung.