Round-Robin-Scheduler in Dafny

Beschreibung des Round-Robin-Scheduler´s:

Wir haben mit Dafny einen Round-Robin-Scheduler (RRS) (1) implementiert und verifiziert. Das Betriebssystem benutzt einen RRS um Prozess-Blöcke, die um eine knappe Ressource konkurrieren, gerecht zu verteilen. Der RRS wird mit Hilfe einer Warteschlange (Queue) nach dem FiFo-Prinzip (First in - First out) realisiert. Die Queue wird mittels einer Sequenz von Prozess-Blöcken implementiert. In der Queue befinden sich die noch zu bearbeitenden Prozess-Blöcke. Der Ablauf des RRS ist wie folgt:
Der erste Prozess-Block wird aus der Queue genommen. Er bekommt sein Quantum an Rechenzeit von der CPU. Wenn das Quantum abgelaufen ist, wird der Prozess am Ende der Queue hinzugefügt. Danach wird ein neuer Prozess aus der Queue genommen. Sollte ein Prozess keine Rechenzeit mehr benötigen, wird er nicht mehr in die Queue eingefügt.

Implementierung des RRS:

Um unseren RRS zu spezifizieren, benötigen wir eine Klasse Prozess-Block (PCB), die wichtige Informationen für das Betriebssystem (OS) speichert. Im folgenden ist der PCB dargestellt.

Der Prozess-Block (PCB):

- speichert seine Prozess-Block-ID (pid) zur eindeutigen Wiedererkennung,
- speichert seine benötigte Rechenzeit (duration),
- speichert die ID vom Prozess-Block-Ersteller (ownerID),
- speichert die bereits erhaltene Zeit von der CPU (usedCPU),
- hat einen Konstruktor „Init(pid: int, duration: int, ownerID: int)“ zur Erstellung eines Prozess-Blockes mit Initialwerten.

Da der PCB im OS verwendet wird, wird nachfolgend das OS mit der Queue als Sequenz von Prozess-Blöcken dargestellt.

Das Betriebssystem (OS):

- speichert eine Sequenz von Prozess-Blöcken (que),
- speichert den aktuellen Prozess-Block (pcb),
- speichert das Quantum (quantum),
- hat das Prädikat „Valid()“. Es überprüft, ob in der Sequenz keine Null-Werte sowie keine doppelten Prozess-Block-ID´s vorkommen,
- hat das Prädikat „inQue(pid: int)“. Es überprüft, ob sich die pid schon in der Sequenz befindet,
- hat einen Konstruktor „Init(quantum: int)“ zum Festlegen des Quantums. Es wird pcb auf „null“ gesetzt. Die Sequenz que wird leer erstellt.
- hat eine Methode „enQueue(prozess: PCB_t)“ zum Hinzufügen eines Prozess-Blockes.

- hat eine Methode „deQueue()“ zum Entnehmen des ersten Elements aus der Queue.
- hat die Methode „operate()“, das die Methode deQueue() aufruft und den PCB in einem lokalen Attribut pcb speichert. Dem Attribut pcb wird bei dem Attribut usedCPU ein Quantum hinzugefügt. Sollte der PCB noch nicht fertig sein, wird er der Queue (que) mit enQueue(pcb) hinzugefügt. Anschließend wird pcb auf null gesetzt.

Die Methode operate():

Die Methode operate() läuft wie oben beschrieben ab. Um diese Methode zu spezifizieren benötigen wir folgende Vorbedingungen, Modifikationen und Nachbedingungen.

Vorbedingungen:
Als Vorbedingung überprüfen wir mit dem Prädikat Valid(), ob in der Queue keine Null-Werte, sowie keine doppelten Prozess-Block-ID´s vorkommen. Eine weitere Vorbedingung ist, dass mindestens ein PCB sich in der Queue befinden muss.

Modifikationen:
Für unsere Spezifizierung geben wir an welche Variablen sich ändern dürfen. Zum einen ändern wir von der Klasse OS die Queue und in der Queue den Inhalt an der Stelle 0, also das erste Element in der Queue.

Nachbedingungen:
Unsere Nachbedingungen der Spezifizierung werden im Folgenden erläutert. Um zu überprüfen, ob die Instanz noch in einem konsistenten Zustand befindet, benutzen wir das Prädikat Valid(). Wir müssen folgende Fallunterschreidung betrachten; es kann sein, dass der aktuelle PCB noch nicht fertig ist. Dann wird er in der Queue hinzugefügt. Sollte der PCB fertig sein, so wird der PCB nicht der Queue hinzugefügt.

Fall a): Der Prozess-Block ist fertig, wenn das Attribut usedCPU plus dem Quantum an Rechenzeit größer gleich dem Attribut duration ist. In diesem Fall wird der PCB der Queue nicht hinzugefügt; Somit müssen folgende Nachbedingungen erfüllt werden:
Es sei ein PCB in der Queue an der Stelle 0 mit folgenden Namen oldPcb benannt.
Die Länge der Queue muss um ein Element kürzer sein als die Länge der alten Queue und die Queue darf keine Null-Werte sowie keine doppelten Prozess-Block-ID´s haben und pid von oldPcb darf in der Queue nicht mehr vorhanden sein und
die Queue ist gleich der alten Queue ohne das erste Element.
Der Quellcode der Nachbedingung sieht wie folgt aus.

```
old (que[0].usedCPU + quantum >= que[0].duration) ==> |que|+1 == old(|que|) && Valid()
&& !inQue(old(que[0].getPid()))&& que==old(que[1..]);
```

Fall b): Der Prozess ist nicht fertig, wenn das Attribut usedCPU plus dem Quantum an Rechenzeit kleiner dem Attribut duration ist. In diesem Fall wird der PCB der Queue hinzugefügt; Somit müssen folgende Nachbedingungen erfüllt werden:
Es sei ein PCB in der Queue an der Stelle 0 mit folgenden Namen oldPcb benannt.
Die Länge der Queue darf sich nicht verändern und
die Queue darf keine Null-Werte sowie keine doppelten Prozess-Block-ID´s haben und in der Queue muss sich die ID von oldPcb befinden und
es gibt einen Prozess-Block pcb der gleich oldPcb ist und am Ende der Queue ist.
Der Quellcode der Nachbedingung sieht wie folgt aus.

old (que[0].usedCPU + quantum < que[0].duration) ==> |que| == old(|que|) && Valid() && inQue(old(que[0].getPid()))&& exists pcb: PCB_t:: (pcb != null && pcb==old(que[0]) && que==old(que[1..]+[pcb]));

Quellen:

1. **Wikipedia.** ***Round-Robin-Scheduler.*** **[Online] 05. 11 2015. https://de.wikipedia.org/wiki/Round_Robin_%28Informatik%29.**