Round-Robin-Scheduler in Dafny

# Beschreibung des Round-Robin-Scheduler´s:

Wir haben mit Dafny einen Round-Robin-Scheduler (RRS)  [[1]](#Q1) implementiert und verifiziert. Der RRS wird mit einer Warteschlange nach dem FiFo-Prinzip (First in - First out) verwaltet. In der Warteschlange befinden sich noch zu bearbeitete Prozesse. Der Ablauf des RRS ist wie folgt. Der erste Prozess wird aus der Warteschlange genommen. Er bekommt sein Quantum an Rechenzeit von der CPU. Wenn das Quantum abgelaufen ist, wird der Prozess am Ende der Warteschlange hinzugefügt. Danach wird ein neuer Prozess aus der Warteschlange genommen. Sollte ein Prozess keine Rechenzeit mehr benötigen, wird er nicht mehr in die Warteschlange eingefügt.

# Implementierung des RRS:

Unsere Implementierung war einen Prozess-Block und ein System (OS) mit einer Queue zu implementieren. Der Aufbau der einzelnen Komponenten sieht wie folgt aus.

Der Prozess (PCB\_t):

* + speichert seine Prozess-ID (pid) zur eindeutigen Wiedererkennung.
  + speichert seine benötigter Rechenzeit (duration).
  + speichert die ID vom Prozessersteller (ownerID).
  + speichert die bereits erhaltene Zeit von der CPU (usedCPU).
  + Hat einen Konstruktor „Init(pid: int, duration: int, ownerID: int)“ zur Erstellung eines Prozesses mit Initialwerten.

Das System (OS):

* + speichert eine Sequenz von Prozessen (que).
  + speichert den aktuellen Prozess (pcb).
  + speichert das Quantum (quantum).
  + hat das Prädikat „Valid()“. Es überprüft, ob es in der Sequenz keine Null-Werte sowie keine doppelten Prozess-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 Prozesses.
  + Hat eine Methode „deQueue()“ zum entnehmen des ersten Elements aus der Sequenz.
  + Hat die Methode „operate()“, dass die Methode deQueue() aufruft und den Prozess in einer lokalen Variable pcb speichert. Der Variablen pcb wird bei der Variable usedCPU ein quantum hinzugefügt. Sollte der Prozess noch nicht fertig sein, wird er der Queue (que) mit enQueue(pcb) hinzugefügt. Anschließend wird pcb auf null gesetzt.

Genauere Betrachtung einer Methode:

Im folgendem werden wir die Methode operate() betrachten, da in dieser Methode die Hauptarbeit erfolgt. Die Methode operate() läuft wie oben beschrieben ab. Damit wir beweisen können, dass die Methode operate() richtig arbeitet, benötigen wir folgende Vorbedingungen, Modifikationen und Nachbedingungen. Spezifieren das verhalten

**Vorbedingungen:**

Mit dem Prädikat Valid() überprüfen wir, ob sich in der Queue keine Null-Werte, sowie keine doppelten Einträge vorkommen. Eine weitere Vorbedingung ist, dass die Länge der Queue größer 0 sein muss, da wir sonst versuchen auf Elemente in der Queue zugreifen wollen, die nicht vorhanden sind und es somit zu einem „Index out of Range“-Fehler kommt.

**Modifikationen:**

Für unsere Beweisführung geben wir an welche Variablen sich ändern. 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 Beweisführung werden im folgendem erläutert. Wir rufen wieder das Prädikat Valid() auf. Desweiteren muss die Länger der Queue größer gleich 0 sein, da es sein kann, dass die Queue keine Elemente mehr hat. Da es in der Methode operate() eine Fallunterscheidung gibt, haben auch bei den Nachbedingungen eine Fallunterscheidung gemacht.

Fall a): Der Prozess ist fertig und wird der Queue nicht hinzugefügt,somit müssen folgende Bedingungen erfüllt werden.

Der alte Prozess in der Queue an der Stelle 0 wird nun mit oldPcb bennant.

Die Variable usedCPU von oldPcb wird mit dem Quantum an Rechenzeit addiert, dieser Wert muss größer gleich der Variable duration von oldPcb sein. Ist dies der Fall wird der Prozess nicht der Queue hinzugefügt. Dann sind folgenden Bedingungen zu erfüllen.

Die Länge der Queue plus 1 muss gleich der Länge der alten Queue sein und  
die Queue darf keine Null-Werte sowie keine doppelten Prozess-ID´s haben und

in der Queue darf sich nicht die ID von oldPcb befindet und

die Queue ist gleich der alten Queue ohne das erste Element in der alten Queue.

Die Nachbedingung wird hier hinzugefügt.

*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 und wird der Queue hinzugefügt, somit müssen folgende Bedingungen erfüllt werden.  
Der alte Prozess in der Queue an der Stelle 0 wird nun mit oldPcb abgekürzt.  
Die Variable usedCPU von oldPcb wird mit dem Quantum an Rechenzeit addiert, dieser Wert muss kleiner der Variable duration von oldPcb sein. Ist dies der Fall wird der Prozess der Queue hinzugefügt. Dann sind folgenden Bedingungen zu erfüllen.  
Die Länge der Queue muss gleich der Länge der alten Queue sein und

die Queue darf keine Null-Werte sowie keine doppelten Prozess-ID´s haben und

in der Queue darf muss sich die ID von oldPcb befindet und

es gibt einen Prozess pcb der ungleich Null ist und er ist gleich oldPcb und die Queue ist gleich der alten Queue ohne dem ersten Element, mit dem Prozess pcb an letzter Stelle.

Die Nachbedingung wird hier hinzugefügt.

*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. *Wikipedia.* [Online] 05. 11 2015. https://de.wikipedia.org/wiki/Round\_Robin\_%28Informatik%29.**