Nebenläufigkeit bei Betriebssystemen
Ein Überblick

Arne Struck

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik, Arbeitsbereich TGI
Proseminar Nebenläufigkeit SS 14

Zusammenfassung. Diese Arbeit soll einen Überblick über Mechanismen zur Verteilung der Rechenleistung an verschiedene Prozesse verschaffen und einige der vorkommenen Probleme aufzeigen. Auf dieser Basis werden auch mögliche Verfahren bei Multiprozessor-Systemen behandelt. Zum Schluss wird dann auf die Echtzeitsysteme eingegangen.

Schlüsselwörter: Scheduling, Multiprozessor-Scheduling, Synchronisation, Echtzeitsysteme

Inhaltsverzeichnis

1 Einleitung

Sobald ein Rechner(system) dazu fähig ist mehr als eine bestimmte Aufgabe auszuführen, können diese Aufgaben in Form von Prozessen und Threads beginnen miteinander um die Systemressourcen zu konkurrieren. Im folgenden werden Threads implizit behandelt, da für sie als leichtgewichtige Prozesse ähnliche Prinzipien gelten.
Der Konkurrenzfall tritt ein, sobald zwei Prozesse zu sich überschneidenden Zeiten rechenbereit sind. Falls nicht ausreichend Ressourcen oder Softwaretechnische Mittel existieren, damit die Prozesse eine zeitgleiche Ausführung erlaubt ist, müssen die Ressourcen aufgeteilt werden. Diese Aufteilung nennt sich Scheduling und ist heutzutage besonders im Falle der Rechenzeit einer CPU relevant, da sie bei einem Rechner das bestimmende Element ist. Das Verfahren nachdem die Ressourcen aufgeteilt werden wird als Scheduling-Strategie bezeichnet.

2 Scheduling - allgemeines

Das allgemeine Ziel von Scheduling sollte sich von selbst erschließen, die Systemressourcen möglichst optimal auszulasten (Balance), aber gleichzeitig keinen Prozess zu benachteiligen (Fairness) wobei zwar je nach Fall sowohl das eine, als auch das andere korrekt sein kann, ist in den meisten Anwendungen letzteres das wichtigere. Es wäre beispielsweise durchaus problematisch, sollten in einem Automobil die Kontrolle der Lichtanlage und der Bremsen durch den selben Mikrocontroller bearbeitet werden und die Lichtanlage vom Scheduler immer eine höhere Priorität eingeräumt bekommen.
Aber natürlich hat jedes System auch seine eigenen Scheduling-Ziele:

Stacksysteme

- Maximierung des Durchsatzes (der geleisteten Arbeit)
- Minimierung der Durchlaufzeit
- Maximierung der CPU-Auslastung

Interaktive Systeme

- Reduktion der Antwortzeit
- Proportionalität gewährleisten (Benutzererwartung erfüllen)

Echtzeitsysteme

- Deadlines einhalten
- Vorhersagbarkeit (Reduktion des Qualitätsverlustes in Multimedia-Systemen)

[Tan09]

Scheduling-Strategien müssen diese Ziele mit ihrer Strategie vereinigen und dadurch ein akzeptables Scheduling erzeugen.

Aber wann muss Scheduling erfolgen? Hier existieren Standartfälle, welche überall vorkommen können und besondere Fälle, welche von der durchzusetzenden Strategie und dem System an sich abhängen. Beispiele für diese Standartfälle wären die Entscheidung ob bei frischem Spawn Eltern- oder Kindprozess weiterlaufen dürfen und die Entscheidung welcher Prozess weiter bearbeitet wird, sollte ein Prozess terminieren oder blockiert (zum Beispiel durch eine Ein- oder Ausgabeanfrage).
Scheduling-Strategien können verdrängend oder nicht verdrängend sein. Nicht verdrängend bedeutet, dass sobald ein Prozess die Priorität bekommt, rechnet er bis zur Blockierung oder Terminierung. Verdrängend wiederum beschreibt die Eigenschaft einer Strategie einen Prozess in seiner Ausführung vom Scheduler unterbrechbar zu machen, sollte ein Prozess mit nach der Scheduling Strategie stärkeren Priorität in der Menge der zu bearbeitenden Prozesse erscheinen. Der verdrängte Prozess wird der Menge zugeschlagen.

3 verschiedene Strategien

3.1 First Come, First Serve

Die First Come, First Serve Strategie ist die simpelste Idee für ein Scheduling und findet auch in anderen Bereichen der Informatik Anwendung. Es handelt sich hierbei um eine nicht verdrängende Strategie. Das Verfahren ist einfach zusammenzufassen: Der erste Prozess, welcher Rechenbereitschaft meldet bekommt die Systemressourcen, der zweite Prozess, welcher Rechenbereitschaft meldet bekommt den ersten Platz in der Warteschlange, der Dritte den zweiten Platz und so weiter.
Dies macht die Strategie einfach zu implementieren und der findige Leser hat auch schon den Vorschlag herauslesen können.

3.2 Shortest Job First

Bei dieser nicht verdrängenden Strategie wird aus der Menge, der zur Ausführung stehenden Prozessen derjenige mit der geringsten Ausführungszeit gewählt. Dies setzt voraus, dass die zu erwartenden Ausführungszeiten bekannt sind. Dies ist auf zwei Arten realistisch zu erreichen: das Programm wurde vom Entwickler auf Laufzeiten getestet, diese müssen dann von allen Entwicklern norminalisiert werden und stellt die Information bereit. Dies ist im großen Rahmen nicht praktikabel. Oder es handelt sich um immer wiederkehrende Prozesse, dann kann der Scheduler über statistische Verfahren eine ungefähre Laufzeit abschätzen.
Allerdings beinhaltet er die Schwachstelle, dass es sich um eine unfaire Strategie handelt, sobald ein Prozessspawn existiert. Dies lässt sich an folgendem Minimalbeispiel nachvollziehen:
Gegeben seien drei Prozesse A, B und C, Prozess A und B haben eine zu erwartende Ausführungszeit von 1 und 2 Prozess C von 20. Nun wird zu erst A,

dann B abgearbeitet. Sollte nun in der Abarbeitungszeit von B ein neuer Prozess A durch B gespawnt werden, wird dieser in die Menge der zu bearbeitenden Prozesse aufgenommen. Sollte nun dieser neue Prozess A einen neuen Prozess B spawnen, befinden wir uns in einer endlosen Schleife, welche die Bearbeitung von C nicht erlaubt. Was den Datendurchsatz angeht ist diese Strategie nachweislich optimal [Tan09]

4 Shortest Remaining Time Next

Bei Shortest Remaining Time Next handelt es sich um eine verdrängende Adaption von Shortest Job First. Anstatt der Gesamtdauer des Prozesses, wird die noch verbleibende Ausführungszeit eines Prozesses berücksichtigt, wobei bei einem frischen Prozess natürlich die komplette zu erwartende Ausführungszeit der verbleibenden entspricht.
Die löst einen großen Teil, der Fairness-Probleme, das Problem aus dem Minimalbeispiel bleibt jedoch bestehen.

4.1 Round Robin

4.2 Priorisiertes Scheduling

4.3 Shortest Process Next

4.4 Multiple Queues

4.5 Guaranteed Scheduling

4.6 Fair-Share-System

5 Multiprozessorsysteme

5.1 Betriebssystem-Aufteilung

1-Kern-1-System-Modell

Master-Slave-Modell

Symmetrisches Modell

5.2 Synchronisation

6 Multiprozessor-Scheduling

7 Echtzeitscheduling

8 Fazit/Ausblick

Literatur

[RHAD14] Andrea C. Arpaci-Dusseau Remzi H. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, Inc., 2014.

[Tan09] A.S. Tanenbaum. *Moderne Betriebssysteme*. Pearson Studium - IT. Pearson Deutschland, 3 edition, 2009.