Kapitel 9: Planung (Scheduling)

- Motivation und Einführung
- CPU Scheduling in Einprozessorsystemen
- CPU Scheduling in Mehrprozessorsystemen (SMPs)
- CPU Scheduling für Echtzeitsysteme

9.1 Motivation und Einführung

Schedulingprobleme und deren Lösungsverfahren begegnen uns im täglichen Leben an vielen Orten, teilweise erwarten wir sie dort, teilweise ist uns dies ohne nähere Kenntnis nicht so ohne weiteres bewusst. Beispielsweise muss ein Koch eines Restaurants täglich zig-Schedulingprobleme beim Zubereiten seiner Menüs lösen, da er wie gesagt in seiner Küche nur über begrenzte Ressourcen wie Töpfe, Pfannen, Kochplatten, Küchenpersonal etc. verfügt, er aber andererseits die an einem Tisch bestellten Menüs möglichst zeitgleich seinen Gästen offerieren möchte.

Eine Reihe von orthogonalen Entwurfsparametern mag bei der Etablierung eines geeigneten Planungsverfahrens (*scheduling*) zu berücksichtigen sein. Prinzipiell kann man sich zunächst die Frage stellen, wie viele CPU-Scheduler sollte man denn überhaupt gleichzeitig in einem System vorsehen. Klassische Lösungen gehen davon aus, dass in einem System genau ein CPU-Scheduler für alle Kernel-Level Threads (bzw. Prozesse) zuständig ist, andererseits gibt es Überlegungen für verschiedene Subsysteme verschiedene Scheduler anzubieten, die -evtl. mit unterschiedlichen Strategien ausgestattet- das Gesamtsystem besser einplanen können als ein einziges zentrales Verfahren.

Mögliche den Schedulingentwurf beeinflussende Entwurfsparameter sind:

- Mit oder ohne Verdrängung
- Unabhängige versus abhängige Prozesse
- Mit oder ohne Berücksichtigung evtl. Interaktionskosten
- Mit und ohne statische/dynamische Prioritäten
- Leistungskriterium

9.1.1 Verdrängende versus nicht verdrängende Umschaltstrategien

Jede Verdrängung eines KLTs (Prozesses) von dem Prozessor sollte bei den aktuellen CPU-Geschwindigkeiten effektiv gerechtfertigt sein, da jeder Verdrängungsvorgang einen erheblichen Aufwand darstellen kann, insbesondere dann, wenn zusätzlich auch der Adressraum umgeschaltet werden muss. Natürlich wird jeder neue bereite bzw. wieder bereite Echtzeitprozess jeden rechenintensiven Hintergrundprozess verdrängen dürfen, denn ansonsten könnte es bei einem rechenintensiven Hintergrundprozess zu lange dauern, ehe der Echtzeitprozess mit seiner zeitkritischen Aktivität beginnen kann.

Analysieren Sie weitere Anwendungsfälle von Verdrängungen.

9.1.2 Unabhängige versus abhängige Prozesse

Manchmal wünscht man sich aus der Sicht des Schedulers Information über wechselseitig abhängige Aktivitäten, z.B. alle KLTs einer Applikation. Je nach System kann man dann die Ablaufplanung des Systems mittels Gangscheduling u.U. beschleunigen.

9.1.3 Berücksichtigung von Interaktionskosten

Unter Umständen können aber auch die IPC-Kosten zwischen den Threads einer multi-threaded Applikation beim Aufstellen des optimalen Ablaufplans in Betracht gezogen werden.

9.1.4 Statische versus dynamische Prioritäten

In vielen Systemen wird mit Prioritäten gearbeitet, mit anderen Worten der Scheduler hat neben seinen sonstigen Planungskriterien auch explizite - vom Benutzer, vom Systemverwalter etc. - zusätzliche Prozessattribute zu berücksichtigen, will er das entsprechende Leistungsziel des Systems erfüllen.

9.1.5 Leistungskriterien

Die wesentlichsten Leistungsmaße, die bei der Güteabwägung eines Schedulingverfahrens eine Rolle spielen, sind:

- Maximale bzw. mittlere Verweildauer (turn around time)
- Maximale bzw. mittlere Antwortzeit
- Durchsatz
- Prozessorauslastung
- Maximale bzw. mittlere Verspätung bzw. Verzögerung

9.2 CPU Scheduling für Einprozessorsysteme

Bei Einprozessorsystemen können die CPU-Scheduler einfacher gestaltet werden, da deren Aktivität nur dann benötigt wird, wenn der gerade rechnende Kernel-Level Thread nicht weiter rechnen kann oder weiter rechnen sollte.

Im ersten Fall wird ein Threadwechsel im Rahmen eines Systemaufrufs (*system call*) durchgeführt, im letzteren Fall kommt es auf Grund einer asynchronen Unterbrechung dazu, dass die aktuelle CPU-Lage vom Scheduler neu zu überdenken ist, evtl. hat der bislang rechnende Kernel-Level Thread bereits lange genug gerechnet, d.h. man will aus Fairnessgründen einen anderen Kernel-Level Thread rechnen lassen. Oder im Zuge einer Peripherieunterbrechung wird ein bislang darauf wartender blockierter Prozess mit hoher Dringlichkeit wieder rechenbereit.

9.2.1 FCFS

Dieses Verfahren zeichnet sich dadurch aus, dass alle Prozesse gleich behandelt werden, niemand wird bevorzugt, ein Prozess nach dem anderen wird auf die CPU zugeordnet und kann dort solange rechnen, bis er entweder terminiert oder blockiert. Jeder neue Prozess wird ans Ende der Bereitwarteschlange eingereiht. Prozesse, die temporär blockiert waren, müssen gemäß ihrer Startzeit in die Bereitwarteschlange eingereiht werden, können also nicht einfach ans Ende der Ready Queue eingereiht werden.

9.2.2 LCFS

Diese Strategie favorisiert indirekt kurze Prozesse, die evtl. fertig werden, bevor der nächste Prozess gestartet wird.

9.2.3 Round-Robin

Diese Strategie baut auf dem Zeitscheibenmechanismus (*time slice*) auf. Frühe Implementierungen dieses Verfahrens operierten mit einer systemweit konstanten Zeitscheibenlänge, eleganter ist es, die Zeitscheibenlänge vom Prozesstyp abhängig zu machen.

9.2.4 Prioritätsscheduling

Das Problem mit Prioritäten in dynamischen Multiprogrammsystemen ist zum einen, wie man den Nutzern die Verwendung von Prioritäten effektiv anbieten sollte, d.h. so, dass nicht jeder Nutzer für alle seine

Prozesse stets die höchste Priorität auswählt. Ein weiteres Problem -zumindest mit statischen Prioritäten- ist die Gefahr, dass weniger wichtige Prozesse verhungern können. Ferner ist damit zu rechnen, dass die so genannte Prioritätsumkehr beobachtet werden kann.

9.2.5 SJN

Diese Strategie setzt entweder eine genaue Kenntnis der virtuellen Prozesszeit auf der CPU voraus, oder kann diese zumindest in etwa abschätzen. Von allen nicht verdrängenden Schedulingstrategien optimiert diese Strategie die mittlere Verweildauer.

9.2.6 Highest Reponse Ratio Next

Damit werden zwar nach wie vor kürzere Prozesse bevorzugt, aber längere Prozesse können nun mit wachsender Response Ratio auch mal wieder die CPU erhalten, so dass deren Diskriminierung nicht ganz so drastisch ausfällt wie bei SJN.

9.2.7 Lotterie Scheduling

Wie bei einer sonstigen Lotterie werden die bevorzugt, die mehr Lose besitzen.

9.2.8 Multi Level Feedback Strategie

Hierbei versucht man dynamisch zu ermitteln, welche Prozesse eher E/A- oder eher rechenintensiv sind. Prinzipiell versucht man die E/A-intensiven Prozesse zu bevorzugen. Initial wird jeder Neuankömmling als E/A-intensiver Thread eingestuft und somit in die wichtigste Bereitwarteschlange eingereiht. Wird die CPU wieder frei, dann wird sie sich bevorzugt den am längsten in der wichtigsten Bereitwarteschlange wartenden bereiten Prozess zuordnen. Sollte diese Liste leer sein, dann wird die CPU versuchen, sich aus der nächst wichtigsten zu bedienen etc.

Sollte ein Prozess seine komplette Zeitscheibe auf der CPU rechnen, dann wird er eine Wichtigkeitsstufe zurückgesetzt und gibt dabei die CPU ab. Prozesse, die wegen E/A blockiert werden, werden wiederum nach Beendigung der E/A in die wichtigste Bereitwarteschlange eingereiht. Man kann unterschiedliche Zeitscheibenlängen vorsehen, je nachdem aus welcher der n Bereitwarteschlangen der neue Prozess zugeordnet wird.

9.3 CPU Scheduling für Mehrprozessorsysteme

Bei mehreren Prozessoren ergibt sich neben der Fragestellung, welcher Prozess beim nächsten freiwerdenden Prozessor gemäß einer Planungsstrategie zugeordnet werden sollte. Man kann sich i.d.T. fragen, ob stets der erste Prozess in der zentralen Bereitwarteschlange (*Ready Queue*) am besten die Schedulingstrategie umsetzen würde.

Wenn man beispielsweise zulässt, dass Programmierer die Möglichkeit haben sollten, insbesondere bei multithreaded Anwendungen lieber alle Kernel-Level Threads auf ein und dem selben Prozessor laufen zu lassen oder ob es besser ist, das durch die m>1 Prozessoren vorhandene Parallelitätsangebot wahrzunehmen und somit die Kernel-Level Threads der gleichen Anwendung möglichst auf viele Prozessoren zu verteilen. Im ersten Fall spricht man bei Windows XP von der so genannten Prozessoraffinität, die ein Attribut einer Applikation sein kann.

9.4 CPU Scheduling für Echtzeitsysteme

Bei Echtzeitsystemanwendungen spielen die so genannten Sollzeitpunkte (*deadlines*) eine Rolle, die einzuhalten sind, da ansonsten die Leistung des Systems als nicht erbracht betrachtet wird. Man unterscheidet zwei verschiedene Echtzeitanwendungen:

1. Harte Echtzeitanwendungen sind solche, deren Sollzeitpunktüberschreitung zu Katastrophen führen (Autopilot, sonstige Kontrollsysteme).
2. Weiche Echtzeitanwendungen sind solche, bei denen eine Beeinträchtigung des Nutzungskomfort zu spüren sind, wenn deren Sollzeitpunkte nicht eingehalten werden, aber keine unmittelbare Gefahr für Leib und Leben entsteht, wenn gelegentliche Sollzeitpunktüberschreitungen auftreten.

9.4.1 EDF

Bei diesem Verfahren werden die Prozesse so in der Bereitwarteschlange angeordnet, dass die mit den knappsten Sollzeitpunkten bevorzugt zugeordnet werden. Dieses Verfahren findet optimiert die maximale Verspätung bzw. Verzögerung und ist ein so genanntes Online-Schedulingverfahren, das also in der Praxis ohne zusätzliche Zugangskontrolle angewendet werden kann.

9.4.2 RMS

Beim ratenmonotonen Scheduling geht man von periodischen Prozessen aus, wobei die Prozesse bevorzugt werden, deren Periodenlänge am kleinsten sind. Damit das Verfahren in der Praxis angewendet werden kann, muss man eine Zugangskontrolle (*admission control*) einsetzen, die offline ausprobiert, ob neue periodische Prozesse zusätzlich ins System eingeschleust werden können.