Duale Hochschule Baden-Württemberg Mannheim

Integrationsseminar

OS Scheduling Algorithmen

Studiengang Wirtschaftsinformatik

Studienrichtung Data Science

Verfasser:	Jannik Völker, Benedikt Prisett, Eric Echtermeyer
Matrikelnummer:	–TODO–, –TODO–, 6373947
Kurs:	WWI-21-DSA
Studiengangsleiter:	Prof. Dr.-Ing. habil. Dennis Pfisterer
Dozent:	Prof. Dr. Maximilian Scherer
Bearbeitungszeitraum:	13.11.2023 – 07.02.2024

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

OS	Betriebssystem
CPU	Central Processing Unit

1 Grundlagen der Visualisierung

Womöglich interessante Elemente: - Farbgebung, welche Farben sollte man wann nehmen, wie etwas darstellen - Bewegung von Objekten, wann sollte sich was wie bewegen. Eher von links nach rechts oder andersrum? - Konventionen einhalten, im Uhrzeigersinn, Titel oben links, ... - Möglichst verwandt aussehen

2 OS Scheduling Algorithmen

Das Kernstück eines jeden modernen Betriebssystems is dessen Fähigkeit eine Vielzahl von Prozessen effizient und effektiv zu verwalten. Diese Prozessverwaltung, auch bekannt als Scheduling, ist eine komplexe Aufgabe, welche darüber entscheidet, welcher Prozess zu welchem Zeitpunkt von der Central Processing Unit (CPU) verarbeitet wird. Da moderne Betriebssysteme stets eine hohe Anzahl von Hintergrundprozessen bis hin zu anspruchsvollen Anwendungen verarbeiten müssen, ist die Verwendung leistungsfähiger OS Scheduling Algorithmen essentiell. Die richtige Auswahl dieser ist ein kritischer Bestandteil, um die Gesamtleistung und Reaktionsfähigkeit des Computersystems zu optimieren. Die zentrale Herausforderung des OS Schedulings ist die begrenzte Leistungsfähigkeit der Hardware, insbesondere der CPU, da diese zu einem Zeitpunkt stets nur einen Prozess ausführen kann. Dieser Engpass wird durch eine intelligente Planung gemindert, bei der entschieden wird, welcher Prozess als nächstes Zugang zur CPU erhalten soll. Ein effektives Scheduling muss dabei eine Balance zwischen verschiedenen wichtigen Metriken finden:

- **Durchsatz (Throughput)**:

  Der Durchsatz misst die Anzahl der Prozesse, die in einer bestimmten Zeiteinheit vollständig abgearbeitet werden. Ein höherer Durchsatz bedeutet eine effizientere Verarbeitung von Prozessen durch das System. Bei vollständiger CPU-Auslastung, ist diese Metrik bei unterschiedlichen Algorithmen konstant.

  $$Durchsatz = \frac{Anzahl\ abgeschlossener\ Prozesse}{Zeiteinheit}$$

  Diese Formel berechnet den Durchsatz als Anzahl der Prozesse, die pro Zeiteinheit abgeschlossen werden.

- **Wartezeit (Waiting Time)**:

  Die Wartezeit ist die Gesamtzeit, die ein Prozess in der Warteschlange verbringt, bevor er Zugang zur CPU erhält. Niedrigere Wartezeiten sind in der Regel wünschenswert, da sie auf ein reaktionsfähigeres System hinweisen.

  $$Wartezeit = \frac{1}{n}\sum_{i=1}^{n}(Startzeit_i - Ankunftszeit_i)$$

Hierbei ist n die Anzahl der Prozesse, und die Wartezeit wird als Durchschnitt der Zeit berechnet, die jeder Prozess vom Ankunftszeitpunkt bis zum Start der Ausführung wartet.

- **Antwortzeit (Response Time)**:

  Die Antwortzeit misst die Zeit vom Beginn eines Prozesses bis zur ersten Antwort, nicht bis zur vollständigen Ausführung. Diese Metrik ist besonders wichtig in interaktiven Systemen, wo eine schnelle Reaktion auf Benutzereingaben erforderlich ist.

  $$Antwortzeit = \frac{1}{n}\sum_{i=1}^{n}(Erste\ Antwortzeit_i - Ankunftszeit_i)$$

  Die Antwortzeit misst die Zeit vom Ankunft bis zur ersten Antwort jedes Prozesses, gemittelt über alle Prozesse.

- **Ausführungszeit (Turnaround Time)**:

  Die Ausführungszeit ist die gesamte Zeit vom Start eines Prozesses bis zu seinem Abschluss. Diese Metrik berücksichtigt sowohl die Wartezeit als auch die Ausführungszeit und gibt somit ein umfassendes Bild von der Effizienz des Scheduling-Algorithmus.

  $$Ausführungszeit = \frac{1}{n}\sum_{i=1}^{n}(Abschlusszeit_i - Ankunftszeit_i)$$

  Die Ausführungszeit wird als durchschnittliche Gesamtzeit berechnet, die ein Prozess vom Ankunftszeitpunkt bis zum Abschluss benötigt.

- **CPU-Auslastung (CPU Utilization)**:

  Diese Metrik gibt an, wie effektiv die CPU genutzt wird. Eine hohe CPU-Auslastung bedeutet, dass die CPU aktiv Prozesse bearbeitet und nicht untätig ist, was auf eine effiziente Nutzung der Ressourcen hinweist.

  $$\text{CPU} - Auslastung = \left(\frac{Gesamtzeit\ \text{CPU} - Beschäftigung}{Gesamtbeobachtungszeit}\right) \times 100\%$$

  Diese Formel gibt den Prozentsatz der Zeit an, in der die CPU aktiv Prozesse bearbeitet hat, bezogen auf die gesamte Beobachtungszeit.

- **Fairness**:

Fairness misst, wie gleichmäßig die CPU-Zeit zwischen den verschiedenen Prozessen aufgeteilt wird. Ein fairer Scheduling-Algorithmus stellt sicher, dass kein Prozess übermäßig bevorzugt oder benachteiligt wird. Es handelt sich hierbei um eine qualitative Metrik und lässt sich nicht direkt durch eine allgemeine Formel quantifizieren. Stattdessen wird sie oft durch die Analyse der Verteilung der CPU-Zeit und der Wartezeiten über die verschiedenen Prozesse hinweg bewertet.

In den folgenden Abschnitten dieses Kapitels werden verschiedene Scheduling-Algorithmen vorgestellt und analysiert. Jeder Algorithmus hat seine eigenen Stärken und Schwächen, die ihn für bestimmte Szenarien und Anforderungen geeignet machen. Von den einfachen, aber grundlegenden Ansätzen wie First Come First Serve bis hin zu komplexeren Strategien wie Multilevel Queue Scheduling, spiegelt die Entwicklung dieser Algorithmen die Fortschritte in der Computertechnologie und unser zunehmendes Verständnis von effizientem Prozessmanagement wider.

2.1 First Come First Serve (FCFS)

First Come First Serve (FCFS), auch bekannt als First In, First Out (FIFO), stellt einen der grundlegendsten Scheduling-Algorithmen in Betriebssystemen dar. Dieser Algorithmus verarbeitet Prozesse in der Reihenfolge ihres Eintreffens, wobei der zuerst ankommende Prozess als erster bedient wird. Die Implementierung von FCFS erfolgt üblicherweise durch eine Warteschlange, in der Prozesse gehalten und sequenziell abgearbeitet werden, sobald die CPU verfügbar ist. Aufgrund seiner Einfachheit und leichten Implementierbarkeit wird FCFS oft in Lehrbüchern wie Silberschatz, Galvin und Gagne (2018) im Kontext grundlegender Betriebssystemkonzepte diskutiert.

Einer der Hauptvorteile von FCFS liegt in seiner Einfachheit und der daraus resultierenden leichten Implementierbarkeit. Dieser Algorithmus ist transparent und vorhersehbar, da die Reihenfolge der Prozessbearbeitung direkt von ihrer Ankunftszeit abhängt. Zudem gewährleistet er eine faire Behandlung der Prozesse ohne Bevorzugung, da jeder Prozess in der Reihenfolge seines Eintreffens bearbeitet wird (Stallings, 2012).

Jedoch weist FCFS auch signifikante Nachteile auf. Ein wesentliches Problem ist der sogenannte Convoy-Effekt, bei dem ein langer Prozess, der früh in der Warteschlange erscheint, nachfolgende, kürzere Prozesse verzögern kann. Diese Situation führt zu einer ineffizienten CPU-Auslastung und verlängerten Wartezeiten, wie von Tanenbaum und Bos (2014)

hervorgehoben. Weiterhin berücksichtigt FCFS weder die Dauer noch die Priorität von Prozessen, was besonders nachteilig für interaktive Systeme ist, in denen schnelle Antwortzeiten entscheidend sind. Diese Mängel machen FCFS für viele moderne Anwendungen, insbesondere in Multitasking-Umgebungen, unpraktikabel.

Insgesamt bietet FCFS einen grundlegenden Einblick in die Konzepte des Prozessscheduling, ist jedoch aufgrund seiner begrenzten Effizienz und Flexibilität in praktischen Anwendungen oft nicht die bevorzugte Wahl.

2.2 Round Robin

Round Robin ist ein weit verbreiteter Scheduling-Algorithmus in Betriebssystemen, der insbesondere für seine Fairness und Eignung für Zeitscheiben-basiertes Multitasking bekannt ist. Dieser Algorithmus weist jedem Prozess in der Warteschlange ein festes Zeitintervall oder Quantum zu. Nach Ablauf des Quantums wird der aktuell laufende Prozess unterbrochen und an das Ende der Warteschlange gestellt, um dem nächsten Prozess in der Warteschlange CPU-Zeit zuzuweisen. Diese Methode gewährleistet, dass alle Prozesse regelmäßige CPU-Zeit erhalten und kein Prozess andere blockiert. Tanenbaum und Bos (2014) heben hervor, dass durch die Rotation aller Prozesse eine gleichmäßigere Verteilung der CPU-Zeit über alle Prozesse erreicht wird.

Ein wesentlicher Vorteil des Round Robin-Algorithmus liegt in seiner Fähigkeit, eine niedrige Antwortzeit für alle Prozesse zu gewährleisten, was ihn besonders für interaktive Systeme geeignet macht. Da jeder Prozess innerhalb eines bestimmten Zeitrahmens bedient wird, wird vermieden, dass Prozesse zu lange auf die CPU-Zuteilung warten müssen. Diese Eigenschaft wird von Silberschatz, Galvin und Gagne (2018) als entscheidend für Systeme mit hoher Prozessanzahl und Anforderungen an die Reaktionsfähigkeit angesehen.

Jedoch hat der Round Robin-Algorithmus auch einige Nachteile. Ein wesentlicher Punkt ist die Wahl der Länge des Zeitquantums. Ein zu kurzes Quantum kann zu häufigen Prozesswechseln führen, was die Overhead-Kosten erhöht und die CPU-Effizienz verringert, wie Stallings (2012) anmerkt. Andererseits kann ein zu langes Quantum die Antwortzeiten für Prozesse verlängern und den Effekt einer quasi FCFS-Behandlung haben. Daher ist die Optimierung des Zeitquantums für die spezifische Anwendungsumgebung kritisch.

Insgesamt bietet der Round Robin-Algorithmus eine ausgewogene Lösung für das Scheduling-Problem, insbesondere in Umgebungen, in denen Fairness und schnelle Antwortzeiten ge-

fordert sind. Seine Einfachheit und Effizienz machen ihn zu einer beliebten Wahl in vielen Betriebssystemen.

2.3 Multilevel Queue Scheduling

Multilevel Queue Scheduling ist ein fortgeschrittener Scheduling-Algorithmus, der in Betriebssystemen verwendet wird, um verschiedene Klassen von Prozessen effizient zu verwalten. Dieser Ansatz teilt die Prozesswarteschlange in mehrere separate Warteschlangen auf, wobei jede Warteschlange eine eigene Prioritätsebene hat. Prozesse werden basierend auf ihren Eigenschaften, wie Prozesspriorität, Prozesstyp oder Speicheranforderungen, in diese Warteschlangen eingeteilt. Silberschatz, Galvin und Gagne (2018) beschreiben, dass jede Warteschlange ihren eigenen Scheduling-Algorithmus haben kann, was eine differenzierte Behandlung der Prozesse ermöglicht.

Ein Hauptvorteil dieses Ansatzes liegt in seiner Flexibilität und Effizienz bei der Behandlung verschiedener Prozesstypen. Beispielsweise können Systemprozesse, interaktive Prozesse und Batch-Prozesse in verschiedenen Warteschlangen mit entsprechenden Prioritäten und Scheduling-Strategien verwaltet werden. Tanenbaum und Bos (2014) betonen, dass durch diese Methode eine bessere Anpassung an die Anforderungen spezifischer Prozesstypen erreicht werden kann, was zu einer verbesserten Gesamtleistung des Systems führt.

Ein Nachteil des Multilevel Queue Scheduling liegt in seiner Komplexität, sowohl in der Implementierung als auch im Management. Die korrekte Einordnung von Prozessen in Warteschlangen und die Auswahl geeigneter Scheduling-Algorithmen für jede Warteschlange erfordern sorgfältige Planung und ständige Anpassung. Wie Stallings (2012) anmerkt, kann dies zu einem erhöhten Overhead führen und die Systemeffizienz beeinträchtigen, wenn nicht richtig verwaltet.

Trotz dieser Herausforderungen bleibt Multilevel Queue Scheduling eine mächtige Methode in Betriebssystemen, insbesondere dort, wo eine Vielzahl unterschiedlicher Prozesse und Anforderungen effizient verwaltet werden muss.

3 Quantitativer Vergleich

4 Anwendungsgebiete

A Quellcode

Der vollständige Quellcode ist über folgenden GitHub Link erreichbar: https://github.com/echtermeyer/OS-Scheduling-Algorithms-Comparison