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 |
| **FCFS** | First-Come-First-Serve |
| **FIFO** | First-In-First-Out |
| **MLQ** | Multilevel-Queue |

# 1 Grundlagen der Visualisierung

Das Ziel dieses Projektes liegt in der Erstellung eines hochwertigen Animationsvideos, um den komplexen Sachverhalt der Scheduling Algorithmen von Betriebssystemen intuitiv darzustellen, sodass interessierte Zuschauer relevante Sachverhalte schnell verstehen können. Um für den Lernenden ein Animationsvideo mit möglichst hoher Qualität erstellen zu können, ist eine Beschäftigung mit theoretischen Grundlagen der Visualisierung und Animation unabdingbar. Daher wird im folgenden auf grundlegende Theorien in einer Literaturrecherche eingegangen, welche direkt mit dem von uns produzierten Animationsvideo eingebracht werden.

Unsere Herangehensweise ein Animationsvideo zu erstellen, wird durch die Dual-Coding-Theorie von Allan Paivio bestätigt, welche besagt, dass Informationen besser verarbeitet und erinnert werden können, wenn diese sowohl visuell als auch verbal präsentiert werden [1]. Diese Theorie wird bei in unserem Animationsvideo umgesetzt, indem neben den visuellen Darstellungen wie Diagrammen oder animierten Algorithmen diese gleichzeitig stets auch auf der Tonspur erklärt werden. Laut der "Cognitive Load Theory"von John Sweller (1988) ist hierbei aber darauf zu achten, dass die kognitive Belastung des Lernenden berücksichtigt wird. Um effektive Bildungsanimationen zu erstellen, ist es entscheidend, die Balance zwischen informativen Inhalten und einer überladenen Darstellung zu finden [2]. Durch die sinnvolle Anwendung von Animationen, die komplexe Konzepte in einfachere visuelle Elemente zerlegen, kann das Verständnis erleichtert und die kognitive Belastung reduziert werden. Um den Lernenden daher kognitiv nicht zu überlasten, ist es essentiell, dass animierte Darstellungen stets passend zu der Tonspur sind und diese beiden Komponenten nicht voneinander abweichen. Sollte dies doch der Fall sein, kann dies zu Verwirrung oder kognitiver Leistungsüberschreitung des Lernenden kommen, wodurch die vermittelten Inhalte nicht aufgenommen werden können.

Um das Animationsvideo einprägsamer für den Lernenden gestalten zu können, ist es sinnvoll die Multimedia-Prinzipien nach Richard E. Mayer (2001) anzuwenden. Diese besagen, dass eine angemessene Kombination von Text, Bildern und Animationen das Lernen deutlich verbessern kann [3]. Insbesondere das Kontiguitätsprinzip, das besagt, dass verbundene Text- und Bildinformationen zeitlich und räumlich nahe präsentiert werden sollten, ist für die Gestaltung von Bildungsanimationen relevant. Dies wird direkt in unserem Animations-

video umgesetzt, wie es in Abbildung 1.1 dargestellt ist. Hier werden mit dezent animiertem Text grafische Vorgänge beschrieben, sodass der Lernende sich diesen Text in Ruhe durchlesen kann ohne bei einer kurzen Unaufmerksamkeit das Verständnis für den gesamten Sachverhalt zu verlieren.

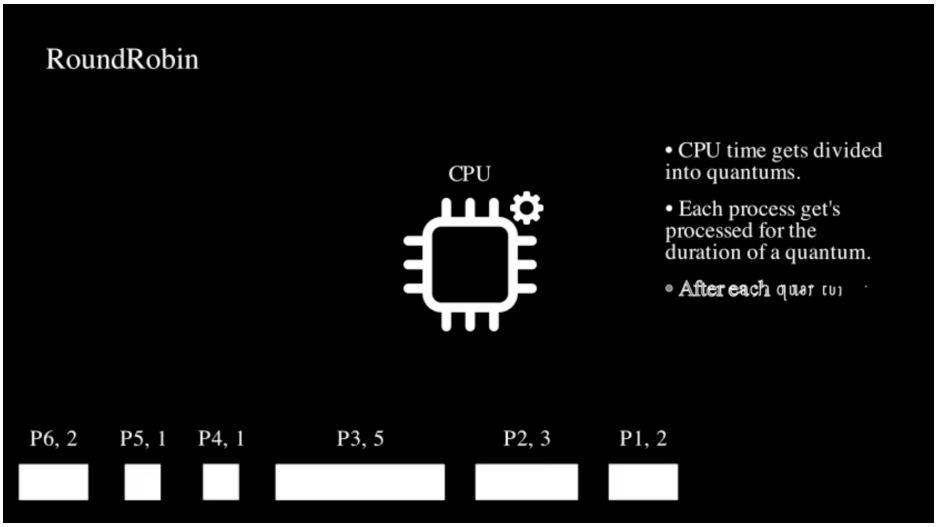


Abbildung 1.1: Die Szene rund um Stelle 00:00min zeigt die Kombination zwischen animierten Bildinformationen mit unterstützendem Text.

Im Kontext von Animationen ist es auch unerlässlich, die Prinzipien der Animation von Disney zu beachten, die von Johnston und Thomas (1981) detailliert beschrieben wurden. Diese Prinzipien, wie SSquash and Stretchünd Änticipation", sind zwar ursprünglich für die Unterhaltungsanimation entwickelt worden, können jedoch auch in Bildungsanimationen angewendet werden, um Konzepte lebendig und verständlich darzustellen. Beispielsweise kann die Bewegung in einer Animation genutzt werden, um die Dynamik eines mathematischen Prozesses zu verdeutlichen.

Neben der Wichtigkeit von Animationen ist eine geschickte Auswahl der vorkommenden Farben ebenso essentiell. Laut der Farbtheorie kann die Auswahl von Farben und Kontrasten dabei helfen, wichtige Informationen zu betonen und die visuelle Erfahrung, und somit die vermittelten Inhalte, der Lernenden zu bereichern [4]. Auch die Gestaltprinzipien der Wahrnehmung, formuliert von Max Wertheimer (1923), bieten ebenfalls wichtige Einsichten für die Gestaltung von Bildungsanimationen [5]. Diese Prinzipien, wie die Gruppierung von Elementen nach Nähe oder Ähnlichkeit, können genutzt werden, um Beziehungen und Strukturen innerhalb der mathematischen Inhalte zu verdeutlichen. So kann beispielsweise das Prinzip der Nähe dazu verwendet werden, um zu zeigen, wie verschiedene mathematische Elemente miteinander in Beziehung stehen.

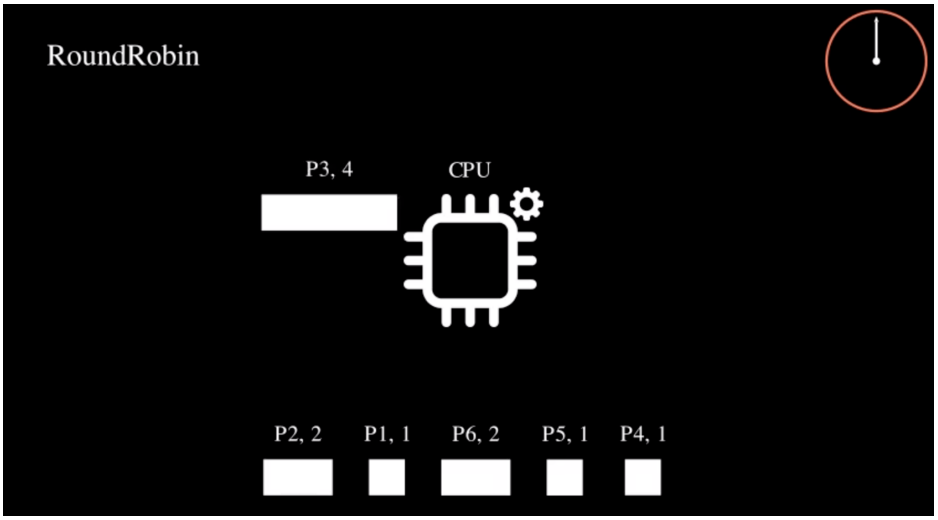


Abbildung 1.2: Die Szene rund um Stelle 00:00min animierte Darstellung des Round Robin Algorithmus, um komplexe Sachverhalte visuell einfach darzustellen.

# 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. Im folgenden werden drei unterschiedliche Algorithmen des OS Scheduling vorgestellt, mit aufsteigender Komplexität. Jeder dieser Algorithmen hat 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

Einer der grundlegenden Scheduling Algorithmen für Betriebssysteme ist First-Come-First-Serve (FCFS), welcher auch als First-In-First-Out (FIFO) bekannt ist. FCFS verarbeitet eingehende Prozesse in der Reihenfolge ihres Eintreffens, wobei der zuerst ankommende Prozess als erstes prozessiert wird. Implementiert wird FCFS für gewöhnlich als Warteschlange, aus welcher eingehende Prozesse anschließend sequentiell verarbeitet werden können.

Der Pseudocode in Abbildung **??** implementiert den FCFS-Scheduling-Algorithmus für einen Satz von Prozessen. Dabei wird angenommen, dass jeder Prozess Eigenschaften wie Ankunftszeit (arrival) und Ausführungszeit (burst) hat. Der Algorithmus berechnet die Startzeit, Endzeit, Wartezeit und Umlaufzeit für jeden Prozess.

Der große Vorteil von FCFS liegt in dessen Einfachheit und der hieraus resultierenden leichten Implementierbarkeit. Daher wird dieser auch oft in Lehrbüchern im Kontext grundlegender Betriebssystemkonzepte diskutiert. Zudem ist FCFS transparent und einfach vor-

hersehbar, da die Reihenfolge und Bearbeitungsdauer aller Prozesse lediglich von deren Ankunftszeiten abhängig ist. Ein zusätzlicher Vorteil liegt in der fairen Behandlung aller Prozesse, welche ohne Bevorzugung stattfindet, da jeder Prozess in der Reihenfolge seines Eintreffens bearbeitet wird.

Nichtsdestotrotz, weist FCFS auch signifikante Nachteile auf, weshalb in der Praxis meist von einer alleinigen Nutzung dieses Algorithmus abgesehen wird. Das wesentliche Problem ist nämlich der Convoy-Effekt, bei dem ein langer Prozess, der früh in der Warteschlange erscheint, nachfolgende, kürzere Prozesse verzögert. Diese Situation führt zu einer ineffizienten CPU-Auslastung und verlängerten Wartezeiten. Weiterhin berücksichtigt FCFS neben der Dauer auch nicht die unterschiedliche Priorität von Prozessen, was besonders nachteilig für interaktive Systeme ist, in denen schnelle Antwortzeiten von höchster Relevanz sind. Diese Mängel machen FCFS für viele moderne Anwendungen unpraktikabel. Daher wird im folgenden der OS Scheduling Algorithmus Round Robin näher betrachtet, welcher versucht eine schnellere Antwortzeit zu ermöglichen.

## 2.2 Round Robin

Round Robin ist ein weit verbreiteter OS Scheduling-Algorithmus in Betriebssystemen, welcher für seine Fairness und Eignung für Zeitscheiben-basiertes Multitasking bekannt ist. Dieser Algorithmus weist jedem Prozess in der Warteschlange ein festes Zeitintervall, auch bezeichnet als 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 zuweisen zu können. Diese Methode gewährleistet, dass alle Prozesse regelmäßige CPU-Zeit erhalten und kein Prozess andere blockiert, wie es bei FCFS mit dem Convoy-Effekt der Fall ist. Durch diese Rotation wird eine gleichmäßigere Verteilung der CPU-Zeit über alle Prozesse erreicht.

Abbildung **??** zeigt Pseudocode für die Implementation des Round Robin Scheduling Algorithmus für einen Satz von Prozessen. Jeder Prozess wird für eine Zeitdauer bis zum definierten Quantum ausgeführt. Der Algorithmus berechnet Start- und Endzeiten, Wartezeiten und Umlaufzeiten für jeden Prozess.

Ein wesentlicher Vorteil des Round Robin-Algorithmus ist dessen Fähigkeit, eine niedrige Antwortzeit für alle Prozesse zu gewährleisten, weshalb dieser besonders für interaktive Systeme von hoher Eignung ist. Da jeder Prozess innerhalb eines bestimmten Zeitrahmens

bedient wird, kommt es nicht zu langen Wartezeiten bis die Prozesse CPU-Zeit zugeteilt bekommen. Diese Eigenschaft wird von Silberschatz, Galvin und Gagne (2018) als entscheidend für Systeme mit hoher Prozessanzahl und Anforderungen an die Reaktionsfähigkeit angesehen, wie es bei modernen Betriebssystemen der Fall ist.

Allerdings weist Round Robin auch Nachteile auf. Ein wesentlicher Punkt ist hierbei die Wahl der Länge des Zeitquantums. Bei einem zu kurzen Quantum, kommt es durch den Round Robin Algorithmus zu häufigen Prozesswechseln, wodurch ein Overhead entsteht und hierdurch die CPU-Effizient verringert wird. Wenn allerdings ein zu langes Quantum gewählt wird, verlängern sich die Antwortzeiten für Prozesse und in extremen Fällen kann es zum gleichen Convoy-Effekt wie bei FCFS kommen. Daher ist die Optimierung des Quantums abhängig von den spezifischen Anwendungsumgebungen und von hoher Relevanz. Ein weiterer Nachteil, ähnlich wie bei FCFS ist die fehlende Betrachtung von unterschiedlichen Prioritäten der eingehenden Prozesse. Im folgenden wird ein weiterer OS Scheduling Algorithmus beschrieben, welcher versucht diese Herausforderung eines prioritäten-basierten Schedulings zu beheben.

Insgesamt bietet der Round Robin-Algorithmus eine ausgewogene Lösung für das Scheduling-Problem, insbesondere in Umgebungen, bei welchen Fairness und schnelle Antwortzeiten gefordert sind. Seine Einfachheit und Effizienz machen ihn zu einer beliebten Wahl in vielen Betriebssystemen.

## 2.3 Multilevel Queue Scheduling

Aufbauen auf anderen Prozess-Schedulern, wie bspw. dem zuvor beschriebenen First Come First Serve oder Round-Robin Prinzip gibt es weitere Verfahren, welche durch eine erweiterte Komplexität beabsichtigen, zuvor dargelegte Prinzipien und Stärken zu vereinen und Schwächen zu überwinden. Eines dieser Verfahren ist das Multilevel Queue Scheduling. Es handelt sich hierbei um einen Algorithmus, bei welchem Prozesse abhängig ihrer Eigenschaften in verschiedene Kategorien eingeteilt und anschließend entsprechend mit unterschiedlicher Priorität bearbeitet werden. Es soll somit durch das Priorisieren von zeitkritischen Aufgaben und der dynamischen Ressourcenzuweisung dem Ziel einer gerechten und effizienten Prozessverwaltung weiter nähergekommen werden. Beim Multilevel Queue Scheduling Verfahren werden zunächst die aktuell offenen, zu bearbeitenden Prozesse unterschiedlichen Warteschlangen dauerhaft zugewiesen. Diese Einteilung kann auf Grundlage

unterschiedlicher Kriterien geschehen und ausschlaggebend können beispielsweise Speichergröße, Prozesspriorität oder der Prozesstyp sein. Eine einfache Aufteilung ist die Zuordnung in Vordergrundaktivitäten und Hintergrundaktivitäten, sodass die eine Gruppe interaktive Prozesse umfasst, welche zeitnah abgearbeitet werden müssen und die andere Gruppe eher statische Prozesse, die ggf. auch länger für die Bearbeitung benötigen. Jede der so geformten Warteschlagen kann unabhängig, auf unterschiedliche Art und Weise bearbeitet werden und verfügt über einen eigenen Scheduling-Algorithmus. So ist es üblich, die Warteschlange für Vordergrundaktivitäten nach dem Round Robin Prinzip abgearbeitet wird, während bei der anderen Warteschlange mit Hintergrundaktivitäten das First Come First Serve Prinzip Anwendung findet. Neben den unterschiedlichen Verfahren, die innerhalb der Wartschlangen stattfinden, gibt es ein einfaches Scheduling-Verfahren zur Verwaltung der Warteschlangen untereinander. Dieses ist für gewöhnlich mit einer festen Priorisierung und präemptiv implementiert. Das bedeutet, dass Warteschlangen absolute Prioritäten über anderen haben und eine höher priorisierte Wartschlange zunächst vollständig abgearbeitet wird, gleichzeitig die Bearbeitung einer niedrig priorisierten Warteschlange aber zugunsten neuer Prozesse in einer anderen Schlange unterbrochen werden kann [6, S.214 f.].

**Algorithmus 1** Multilevel-Queue-Scheduling

1: **Initialize:** $n$ Warteschlangen $Q_1, Q_2, \ldots, Q_n$, jede mit ihrem eigenen Scheduling-Algorithmus $S_1, S_2, \ldots, S_n$
2: Zuweisung von jedem Prozess, basierend auf seinem Typ oder seiner Priorität, in eine Warteschlange
3: **while** es noch zu abzuarbeitende Prozesse gibt **do**
4:  **for** jede Warteschlange $Q_i$ in Prioritätsreihenfolge **do**
5:  **if** $Q_i$ nicht leer ist **then**
6:  Wähle einen Prozess $P$ aus $Q_i$ mithilfe des Scheduling-Algorithmus $S_i$
7:  Führe Prozess $P$ aus
8:  **if** Prozess $P$ abgeschlossen ist **then**
9:  Entferne $P$ aus $Q_i$
10:  **else if** ein Prozess mit höherer Priorität eintrifft **then**
11:  Unterbreche $P$ und füge ihn zurück in $Q_i$ ein
12:  **end if**
13:  **end if**
14:  **end for**
15: **end while**

Der Algorithmus in 1 stellt dar, wie das MLQ-Verfahren im Allgemeinen mit einer beliebigen Anzahl an Warteschlangen abläuft. In einem vereinfachen Beispiel mit zwei Warteschlangen würden zunächst die Vordergrundaktivitäten nach dem Round Robin Verfahren

abgearbeitet und sobald diese Wartschlange leer ist, die Hintergrundaktivitäten mittels First Come First Serve erledigt werden. Tritt während der Bearbeitung der Hintergrundaktivitäten ein Vordergrundprozess auf, wird diese Bearbeitung solange unterbrochen, bis wieder alle Vordergrundaktivitäten abgearbeitet sind. Das Verfahren ist hierbei nicht wie in diesem Beispiel auf zwei Warteschlangen beschränkt, sondern kann um eine Vielzahl an Schlangen für verschiedene Prozesseigenschaften erweitert werden, wie in Abbildung 2.1 dargestellt.

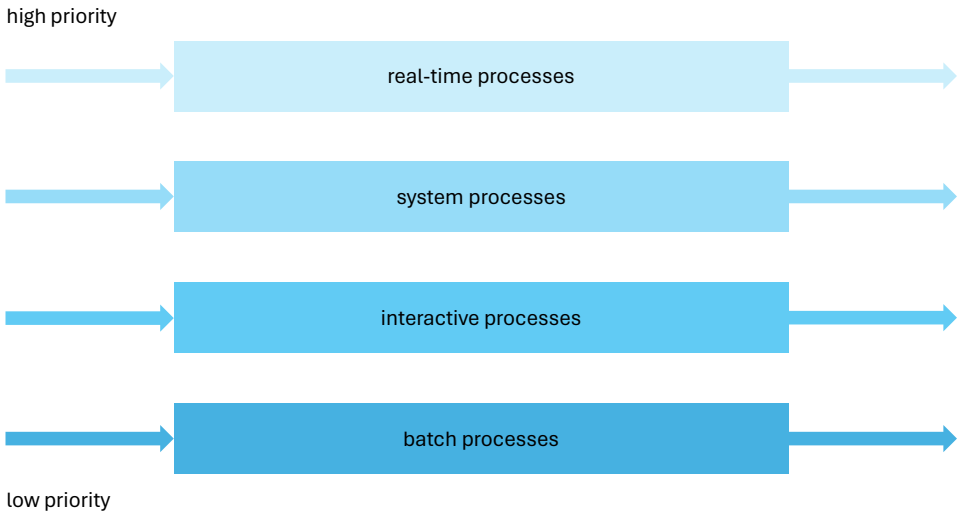


Abbildung 2.1: Darstellung des MLQ-Schedulings mit mehreren Warteschlangen [6, S.215]

Das Multilevel Queue Scheduling Verfahren bietet aufgrund seiner erweiterten Komplexität gegenüber herkömmlichen deutlich einfacheren Verfahren einige Vorteile, aber auch Nachteile. So ist positiv zu bemerken, dass die Reaktionszeit des Systems durch die effizientere Ressourcenallokation reduziert werden kann und die Nutzererfahrung durch die schnellere Abarbeitung von interaktiven Prozessen verbessert wird. Des Weiterem kann mit diesem Verfahren unter den richtigen Umständen der Durchsatz gesteigert werden und das System ggf. auch Prozesse unterschiedlicher Schlangen gleichzeitig ausführen, welches zu der Effizienz des Systems beiträgt. Negativ zu betrachten sind hingegen die erhöhte Komplexität beim Design eines effizienten Systems sowie der zusätzliche Arbeitsaufwand zur Verwaltung der mehreren Warteschlangen untereinander, welcher die Performance des Systems wiederum lindern kann. Ein weiteres Problem ist das „Verhungern“ von Prozessen in einer

Warteschlange mit niedrigerer Priorität, welches auftreten kann, wenn zu viele, große Prozesse in anderen Schlangen zuerst abgearbeitet werden müssen. Um diesem Nachteilen des „Verhungerns“ von Prozessen entgegenzuwirken, gibt es Weiterentwicklungen der einfachen Multilevel Queue wie beispielsweise das heute verbreitete Multilevel Feedback Queue Verfahren. Hier sind die Prozesse nicht komplett fest einer Warteschlange zugeordnet, sondern können auch zu einem späteren Zeitpunkt noch in eine andere Schlange verschoben werden. Dies hilft dabei, dass das System besser auf die Bearbeitungszeiten der Prozesse reagieren kann und keine Prozesse vernachlässigt werden.

# 3 Performance Analyse

Die richtige Auswahl der zuvor vorgestellten Scheduling Algorithmen ist ein essenzieller Bestandteil, um die Gesamtleistung und Reaktionsfähigkeit des Systems zu optimieren. Die zentrale Herausforderung hierbei ist die begrenzte Verfügbarkeit der CPU, da diese zu einem Zeitpunkt stets nur einen Prozess ausführen kann. Während OS Scheduling Algorithmen versuchen den Einsatz der CPU zu optimieren, indem diese entscheiden welcher Prozess als nächstes bearbeitet werden soll, muss stets der am besten geeignete Algorithmus ausgewählt werden [7]. Daher wird sich in diesem Kapitel mit den wichtigsten Metriken zur Auswahl der Algorithmen beschäftigt, um im anschließenden Abschnitt FCFS, Round Robin und MLQ basierend auf Simulationsergebnissen miteinander zu vergleichen.

## 3.1 Metriken

Um unterschiedliche Scheduling Algorithmen wie FCFS, Round Robin oder MLQ bezüglich ihrer Performance und somit ihrer Eignung für Anwendungsgebiete vergleichen zu können, ist es möglich zahlreiche diverse Metriken zu verwenden. Die Auswahl ist abhängig vom jeweiligen Anwendungsgebiet, beispielsweise wird in interaktiven Systemen eine geringe Antwortzeit priorisiert, während bei Batch-Processing-Systemen eine niedrige Ausführungszeit von höherer Relevanz ist [8]. Im Folgenden werden häufig genutzte Metriken erklärt, welche im Anschluss für den quantitativen Vergleich verwendet werden.

**CPU-Auslastung (CPU Utilization)**: Diese Metrik gibt an, wie effektiv die CPU genutzt wird. Eine hohe CPU-Auslastung bedeutet, dass die CPU aktiv an Prozessen arbeitet, was ein Indikator auf eine effiziente Nutzung der Ressourcen ist [9]. Die folgende Formel 3.1 berechnet den prozentualen Wert der CPU-Auslastung, zu welcher die CPU Prozesse bearbeitet, bezogen auf die gesamte Beobachtungszeit.

$$\textit{CPU-Auslastung} = \frac{\textit{Gesamtaktive Zeit}}{\textit{Gesamtbeobachtungszeit}} \times 100\% \tag{3.1}$$

**Durchsatz (Throughput)**: Der Durchsatz misst die Anzahl der Prozesse, welche in einer zuvor definierten Zeiteinheit vollständig abgearbeitet werden. Hierbei bedeutet ein höherer

Durchsatz eine effizientere Verarbeitung von Prozessen durch das System [9]. Bei vollständiger CPU-Auslastung ist der Durchsatz bei unterschiedlichen Algorithmen konstant, sofern von einem möglichen Overhead durch Context Switches abgesehen wird. Die Formel 3.2 berechnet den Durchsatz als Anzahl der Prozesse, die pro Zeiteinheit abgeschlossen werden.

$$\textit{Durchsatz} = \frac{\textit{Anzahl der Prozesse}}{\textit{Zeiteinheit}} \tag{3.2}$$

**Antwortzeit (Response Time)**: Die Antwortzeit berechnet die Zeit vom Beginn eines Prozesses bis zur ersten Antwort. Diese Metrik ist besonders wichtig in interaktiven Systemen, wo eine schnelle Reaktion auf Benutzereingaben erforderlich ist [9]. In Formel 3.3 wird die Berechnung der Wartezeit dargestellt, welche die durchschnittliche Antwortzeit über alle $n$ Prozesse ist.

$$\textit{Antwortzeit} = \frac{\sum_{i=1}^{n} \textit{Antwortzeit Prozess}_i}{n} \tag{3.3}$$

**Wartezeit (Waiting Time)**: Die Wartezeit eines Prozesses ist die Gesamtzeit, welche dieser in der Warteschlange verbringt, bevor er Zugang zur CPU erhält [9]. Niedrigere Wartezeiten sind meist erstrebenswert, da es sich hierdurch für gewöhnlich um ein reaktionsfähigeres System handelt. In folgender Formel 3.4 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.

$$\textit{Wartezeit} = \frac{\sum_{i=1}^{n} \textit{Wartezeit Prozess}_i}{n} \tag{3.4}$$

**Ausführungszeit (Turnaround Time)**: Bei der Ausführungszeit handelt es sich um die gesamte Dauer vom Start eines Prozesses bis zu dessen Abschluss [9]. Es wird somit sowohl die Wartezeit, als auch die Bearbeitungszeit berücksichtigt, weshalb ein umfassenderer Vergleich bezüglich der Effizienz von Scheduling Algorithmen möglich ist. Die Ausführungszeit wird als durchschnittliche Gesamtzeit berechnet, die ein Prozess vom Ankunftszeitpunkt bis zum Abschluss benötigt.

$$\textit{Ausführungszeit} = \frac{\sum_{i=1}^{n} \textit{Ausführungszeit Prozess}_i}{n} \tag{3.5}$$

**Fairness**: Die Metrik der 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 [10]. Es handelt sich hierbei um eine qualitative Metrik, weshalb es in der Literatur keine festgelegte Berechnung hierfür gibt. Stattdessen wird die Fairness oft durch die Analyse der Verteilung von den Wartezeiten über die verschiedenen Prozesse hinweg bewertet, beispielsweise durch die Berechnung der Standardabweichung. Folgende Formel 3.6 stellt die beispielhafte Berechnung hiervon dar, wobei dies nicht als universeller Weg angesehen wird.

$$\textit{Fairness} = \sqrt{\frac{\sum_{i=1}^{n}(\textit{Wartezeit Prozess}_i - \textit{durchschnittliche Wartezeit})^2}{n}} \tag{3.6}$$

## 3.2 Simulationsergebnisse

- Erst kurz den Aufbau beschreiben. - Nutze einfach gleich so 10k an Prozesse und erkläre wie diese Prozesse erstellt wurden –> damit es nachvollziehbar sind. - Dann kann ich erst eine Abbildung der Boxplots mitaufnehmen und zusätzlich noch eine Tabelle wo alles drinnen steht. - Dann erklären was die bedeuten und was für Insights man ziehen kann. - Am Ende noch eine Bestätigung, dass das mit der allgemeinen Literatur übereinstimmt

# 4 Anwendungsgebiete

Innerhalb dieses Kapitels werden wird auf konkrete Anwendungsgebiete der zuvor genannten OS-Scheduling Algorithmen eingegangen. Da diese Algorithmen auch außerhalb von Betriebssystemen einen zentrale Rolle spielen, werden auch Beispiele solcher Anwendungsgebiete aufgenommen.

## 4.1 Innerhalb von Betriebssystemen

Ein häufiges Anwendungsgebiet von FCFS ist für Aufgaben der Batch-Verarbeitung, da hierbei der Convoy-Effekt nicht entscheidend ist. Ein klassisches Beispiel ist das Druckmanagement in frühen Betriebssystemen, wo Druckaufträge in der Reihenfolge ihres Eintreffens abgearbeitet werden.

Round Robin hingegen ist ein weit verbreiteter Scheduling Algorithmus in zeitkritischen Betriebssystemen. Ein Beispiel hierfür ist das Thread-Scheduling in Unix-basierten Systemen, bei dem jedem Thread eine feste Zeitscheibe zugeteilt wird.

MLQ wird in modernen Betriebssystemen wie Linux angewendet, um Prozesse basierend auf ihrer Priorität zu ordnen, wobei systemkritische Prozesse höher priorisiert werden als Benutzerprozesse.

## 4.2 Außerhalb von Betriebssystemen

FCFS wird in zahlreichen Gebieten verwendet, insbesondere bei Prozessen die eine schwierige Digitalisierbarkeit aufweisen. Beispielsweise wird FCFS in der Industrieautomatisierung, insbesondere in der Steuerung von Fertigungsstraßen, verwendet, bei welcher Aufträge in der Reihenfolge ihres Eingangs bearbeitet werden. Auch in Bereichen fernab von Computern und Industrie wird FCFS in alltäglichen Situationen eingesetzt. Ob die Warteschlange an der Kasse im Supermarkt, die Patientenabfertigung im Krankenhaus in der Notaufnahme oder auch der Essensausgabe in der Kantine.

Ein bekanntes Anwendungsgebiet von Round Robin ist die Telekommunikation. Hierbei wird Round Robin für die Paketvermittlung und Lastverteilung in Netzwerkroutern eingesetzt. Ein Beispiel ist die Verteilung von Netzwerkbandbreite in Cisco-Routern. Darüber hinaus wird Round Robin in der Luftfahrtindustrie für die Flugzeugbodenabfertigung eingesetzt, um eine faire und effiziente Zuteilung von Abfertigungsdiensten zu gewährleisten.

MLQ findet beispielsweise Anwendung in Cloud-Computing-Umgebungen wie AWS oder Google Cloud, wo verschiedene Instanzen oder Services in getrennten Warteschlangen basierend auf SLAs verwaltet werden.

# A Quellcode

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

# Literaturverzeichnis


[1] Allan Paivio. „Dual coding theory: Retrospect and current status“. In: *Canadian Journal of Psychology / Revue canadienne de psychologie* 45.3 (1991). Place: Canada Publisher: Canadian Psychological Association, S. 255–287. ISSN: 0008-4255. DOI: 10.1037/h0084295.

[2] John Sweller. „Cognitive Load Theory“. In: *Psychology of Learning and Motivation*. Hrsg. von Jose P. Mestre und Brian H. Ross. Bd. 55. Academic Press, 1. Jan. 2011, S. 37–76. DOI: 10.1016/B978-0-12-387691-1.00002-8. URL: https://www.sciencedirect.com/science/article/pii/B9780123876911000028 (besucht am 03. 01. 2024).

[3] Richard E. Mayer. „Multimedia learning“. In: *Psychology of Learning and Motivation*. Bd. 41. Academic Press, 1. Jan. 2002, S. 85–139. DOI: 10.1016/S0079-7421(02)80005-6. URL: https://www.sciencedirect.com/science/article/pii/S0079742102800056 (besucht am 03. 01. 2024).

[4] Louise Ballard. „The Art of Color by Johannes Itten“. In: *The Journal of Aesthetics and Art Criticism* 22.3 (1. März 1964), S. 344. ISSN: 0021-8529. DOI: 10.2307/427243. URL: https://doi.org/10.2307/427243 (besucht am 03. 01. 2024).

[5] Max Wertheimer. „Untersuchungen zur Lehre von der Gestalt“. In: *Gestalt Theory* 39 (1. März 2017). DOI: 10.1515/gth-2017-0007.

[6] Abraham Silberschatz, Peter B. Galvin und Greg Gagne. *Operating system concepts*. Global edition, [Tenth edition]. Hoboken, NJ: Wiley, 2019. ISBN: 9781119454083.

[7] Neetu Goel und R. B. Garg. *A Comparative Study of CPU Scheduling Algorithms*. 16. Juli 2013. DOI: 10.48550/arXiv.1307.4165. arXiv: 1307.4165[cs]. URL: http://arxiv.org/abs/1307.4165 (besucht am 10. 01. 2024).

[8] Malhar Thombare et al. „Efficient implementation of Multilevel Feedback Queue Scheduling“. In: *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). März 2016, S. 1950–1954. DOI: 10.1109/WiSPNET.2016.7566483. URL: https://ieeexplore.ieee.org/abstract/document/7566483 (besucht am 10. 01. 2024).

[9] Sajeewa Pemasinghe und Samantha Rajapaksha. „Comparison of CPU Scheduling Algorithms: FCFS, SJF, SRTF, Round Robin, Priority Based, and Multilevel Queuing“. In: *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*. 2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC). ISSN: 2572-7621. Sep. 2022, S. 318–323. DOI: 10.1109/R10-HTC54060.2022.9929533. URL: https://ieeexplore.ieee.org/abstract/document/9929533 (besucht am 10. 01. 2024).

[10] S. Haldar und D. K. Subramanian. „Fairness in processor scheduling in time sharing systems“. In: *ACM SIGOPS Operating Systems Review* 25.1 (2. Jan. 1991), S. 4–18. ISSN: 0163-5980. DOI: 10.1145/122140.122141. URL: https://dl.acm.org/doi/10.1145/122140.122141 (besucht am 10. 01. 2024).