Duale Hochschule Baden-Württemberg Mannheim

Integrationsseminar

OS Scheduling Algorithmen

Studiengang Wirtschaftsinformatik

Studienrichtung Data Science

Verfasser:	Eric Echtermeyer, Benedikt Prisett, Jannik Völker
Matrikelnummer:	6373947, , 5370226
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
CFS	Completely Fair Scheduler
EEVDF	Earliest Eligible Virtual Deadline First

1 Grundlagen der Visualisierung

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

Unsere Herangehensweise ein Lehrvideo 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 aufbereitet werden [1]. Dies ist ein Kernaspekt von Lehrvideos und wird umgestzt, indem neben visuellen Darstellungen wie Diagrammen oder animierten Algorithmen diese stets zugleich auf der Tonspur erklärt werden. Laut der Cognitive Load Theorie von John Sweller ist hierbei allerdings darauf zu achten, dass die kognitive Belastung des Lernenden berücksichtigt wird. Um ein effektives Lehrvideo zu erstellen, ist es entscheidend, eine Ausgewogenheit zwischen informativen Inhalten und überladenen Darstellungen zu finden [2]. Animationen können hierbei helfen komplexe Konzepte in einfach zu verstehende visuelle Elemente zu zerlegen, wodurch die kognitive Belastung reduziert werden kann. Um den Lernenden hierbei jedoch nicht kognitiv zu überlasten, ist es essentiell, dass Animationen stets passend und synchron zu der Tonspur angeordnet sind und diese beiden Komponenten nicht voneinander abweichen. Sollte dies doch der Fall sein, kann Verwirrtheit oder eine kognitiver Leistungsüberschreitung des Lernenden entstehen, wodurch die vermittelten Inhalte nicht zielgerichtet aufgenommen werden können [2].

Eine Möglichkeit diese kognitive Überlastung zu vermeiden, ist die Anwendung der Multimedia Prinzipien von Richard E. Mayer. Dieser erarbeitete, dass eine angemessene Kombination von Text, Bildern und Animationen das Lernen deutlich verbessern kann, da Lehrinhalte hierduch prägnanter dargestellt werden können [3]. Insbesondere das Kontiguitätsprinzip, welches besagt, dass verbundene Text- und Bildinformationen zeitlich und räumlich nahe präsentiert werden sollten, ist für die Gestaltung von Bildungsanimationen relevant. Durch

einen dezent animierten Text können grafische Vorgänge parallel zur Tonspur beschrieben werden. Der Lernende kann sich diesen Text nun im eigenen Tempo durchlesen ohne bei einer kurzen Unaufmerksamkeit das Verständnis für den gesamten Sachverhalt zu verlieren. Da es sich hierbei um ein zentrales Prinzip handelt, wird dieses innerhalb unseres Lehrvideos ebenfalls zahlreich umgesetzt.

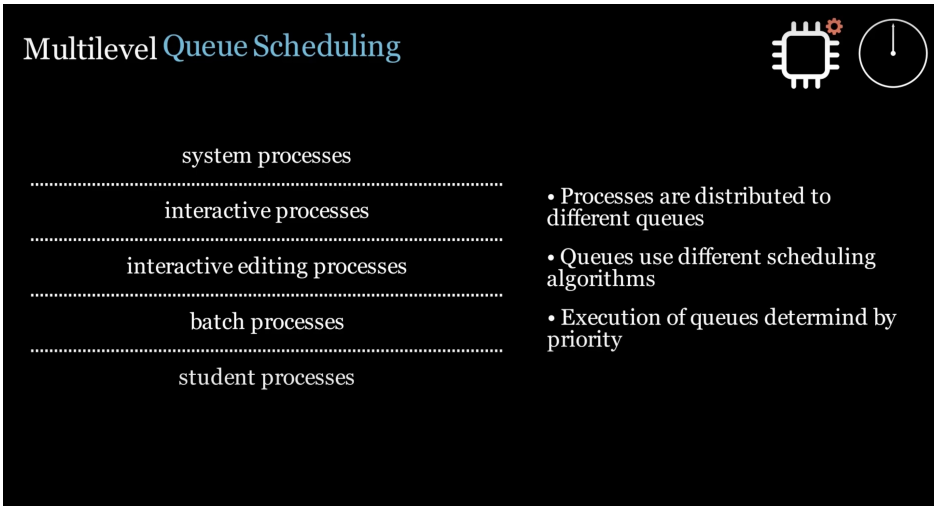

Abbildung 1.1: Die Szene zur Erklärung des MLQ-Algorithmus zeigt die Kombination zwischen animierten Bildinformationen mit unterstützendem Text.

Des Weiteren bietet die Gestaltpsychologie von Kurt Koffka wertvolle Einsichten zur visuellen Wahrnehmung, insbesondere wie Menschen visuelle Informationen verarbeiten und interpretieren [4]. So spielt die Hierarchie visueller Elemente eine entscheidende Rolle, indem diese das Hervorheben wichtiger Inhalte ermöglicht und die Aufmerksamkeit des Lernenden gezielt lenken kann. Die Führung des Blicks durch eine durchdachte Komposition und die strategische Verwendung von Leerflächen können ebenfalls dazu beitragen, die kognitive Belastung zu minimieren und die Aufnahme der Lerninhalte zu erleichtern [4]. Indem diese beispielhaft genannten Prinzipien berücksichtigt werden, können Lehrvideos nicht nur ästhetsich ansprechend sein, sondern auch die Lernprozesse durch eine verbesserte visuelle Wahrnehmung unterstützen. Beispielsweise werden die Funktionsweisen der unterschiedlichen Scheduling Algorithmen innerhalb unseres Lehrvideos durch sich bewegende Prozesse dargestellt, welche durch eine sich drehende CPU abgearbeitet werden. Dies unterstützt dem Zuschauer Zusammenhänge leichter und schneller zu verstehen.

Auch die Gestaltprinzipien der Wahrnehmung von Max Wertheimer, welche von der Gestaltpsychologie abzugrenzen sind, bieten 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 zu verdeutlichen. So kann beispielsweise das Prinzip der Nähe dazu verwendet werden, um zu zeigen, wie verschiedene Elemente miteinander in Beziehung stehen.

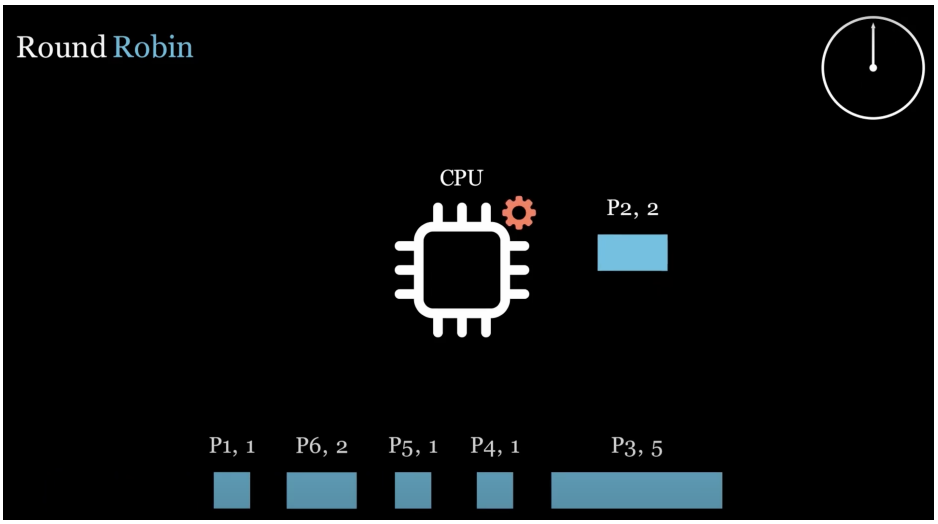

Abbildung 1.2: Wird ein Prozess von der CPU abgearbeitet, reduziert sich die Breite des Prozesses, um eine Verringerung der verbleibenden Bearbeitungszeit darzustellen.

Neben der Wichtigkeit von Animationen und Komposition ist eine geschickte Auswahl der gewählten Farben ebenso essentiell. Laut der Farbtheorie kann die Auswahl von Farben und Kontrasten dabei unterstützen, wichtige Informationen hervorzuheben und die visuelle Erfahrung, und somit die vermittelten Inhalte, der Lernenden zu bereichern [6]. Das von uns erstellte Lehrvideo versucht eine konsistente farbliche Harmonie zu bewirken, was durch die kontinuierliche Verwendung einer abgestimmten Farbpalette erreicht wird. Teilweise werden zudem wichtige Komponenten mithilfe von Sekundärfarben hervorgehoben, wie beispielsweise in Abbildung 1.3 das Erscheinen eines neuen Prozesses durch die Farbe orange betont wird.

Abgesehen der zuvor beschriebenen visuellen Komponenten nimmt die Musik ebenfalls eine zentrale Rolle ein, da diese entscheidend zur Steigerung der Aufmerksamkeit, zur Steuerung der Stimmung und zur Verbesserung des Verständnis beitragen kann. John A. Sloboda betont in "The Musical Mind: The Cognitive Psychology of Music", dass Musik und Soundeffekte nicht nur die emotionale Reaktion der Zuschauer beeinflussen, sondern auch deren Engagement und Erinnerungsvermögen fördern können [7]. Durch den gezielten Einsatz von Musik können bestimmte Stimmungen erzeugt oder verstärkt werden, was die Aufnahme und Verarbeitung von Informationen unterstützt. Ebenso können Soundeffekte

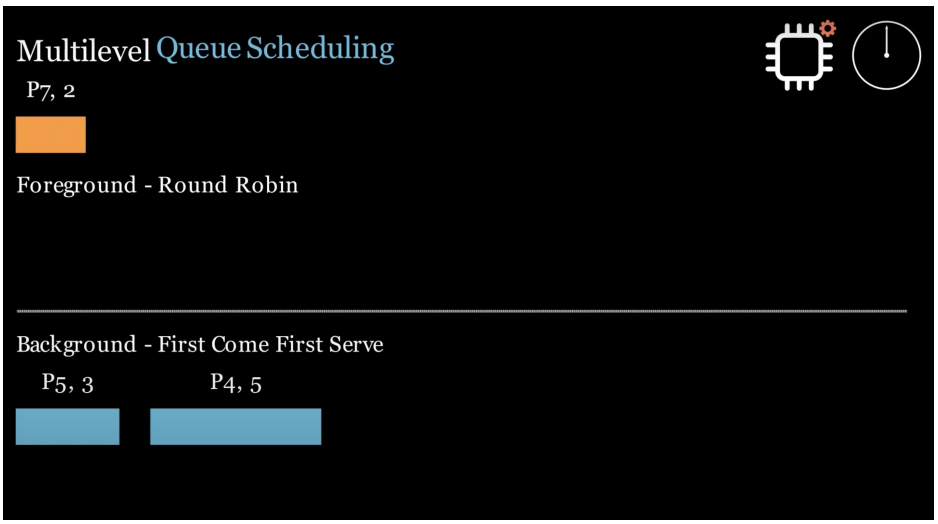

Abbildung 1.3: In der MLQ Szene wird ein neu hinzukommender Prozess mithilfe einer orangenen Farbe für den Lernenden hervorgehoben.

dazu dienen, wichtige Momente hervorzuheben und den Lernenden dazu anzuregen, ihre volle Aufmerksamkeit auf spezifische Aspekte des Videos zu richten [7]. So versucht unser Lehrvideo mit einer ansprechenden und beruhigenden audiovisellen Erlebnis die kognitive Verarbeitungsfähigkeit der Lernenden zu fördern und ein effektiveres Lernerlebnis zu bieten.

Nachdem nun visuelle und auditive Aspekte beschrieben wurden, ist die Erwähnung anderer Aspekte wie der Integration von Storytelling innerhalb des Lehrvideos essentiell. Auch wenn es sich hierbei nicht um eine Art der Visualisierung handelt, kann hierduch eine eine emotionale Verbindung zum Lehrmaterial hergestellt werden, worduch dieses zugänglicher und einprägsamer wird. Roger C. Schank unterstreicht in "Tell me a story: A new look at real and artificial memory"die Relevanz von Geschichten auf das menschliche Verständnis und Erinnerungsvermögen [8]. Indem Lehrvideos eine solche narrative Stringenz aufweisen, bei welcher beispielsweise die Optimierung einer ToDo-Liste durch den Einsatz eines Betriebssystem-Scheduling-Algorithmus als durchgängige Story dient, kann das Interesse und die Motivation der Zuschauer gesteigert werden. Die in unserem Lehrvideo dargestellte ToDo-Liste enthält hierbei bewusst Einträge, welche eine persönliche Relevanz für den Zuschauer haben, um eine emotionale Verbindung herzustellen. Durch diesen Ansatz wird zu Beginn und am Ende des Videos nicht nur die Aufmerksamkeit gefesselt, sondern es wird auch ein Rahmen geschaffen, der das Verständnis und die Merkfähigkeit der vermittelten Inhalte maßgeblich verbessert.

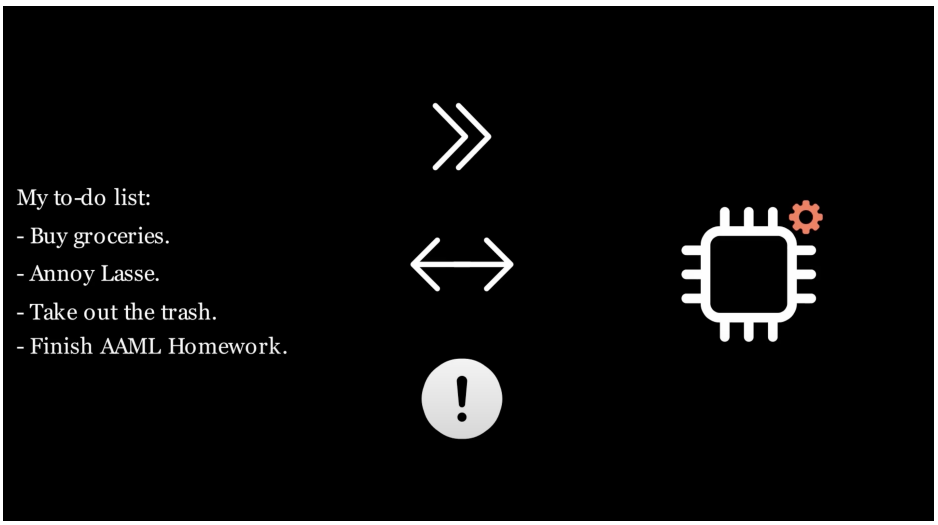

Abbildung 1.4: Die ToDo Liste dient als Einleitung und Abschluss des Lehrvideos, um eine inhaltliche Stringenz zu bewirken.

Nachdem nun auf die theoretischen Grundlagen der Visualisierung eingegangen wurde, beschäftigen sich die folgenden Kapitel mit der Erklärung verwendeter Scheduling Algorithmen und einer anschließenden Performance Analyse dieser im Vergleich.

2 OS Scheduling Algorithmen

Das Kernstück eines jeden modernen Betriebssystems ist 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 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), auch bekannt als First-In-First-Out (FIFO). FCFS verarbeitet eingehende Prozesse in der Reihenfolge ihres Eintreffens, wobei der zuerst eingetroffene Prozess zuerst abgearbeitet wird. Ein Prozess wird so lange von der CPU bearbeitet, bis er vollständig abgearbeitet ist. Dadurch wird jedoch verhindert, dass in der Zwischenzeit andere, wichtigere Prozesse bearbeitet werden [9, Kapitel 2.3.1.7].

In der Regel wird FCFS als Warteschlange implementiert, von der aus eingehende Prozesse sequentiell abgearbeitet werden können.

Das Beispiel in Algorithmus 1 implementiert den FCFS-Scheduling-Algorithmus. Die Prozesse werden in einer Warteschlange Q gespeichert und sequentiell abgearbeitet. Der Algorithmus weist der CPU den ersten Prozess in der Warteschlange zu und bearbeitet diesen, bis die Restlaufzeit des Prozesses abgelaufen ist. Danach wird der nächste Prozess aus der

Algorithmus 1 First Come First Serve

```
1: Initialize: Prozesswarteschlange Q
2: while True do
3:     Prozess P ← Q.dequeue()
4:     Weise CPU P für Zeit P.Restlaufzeit zu
5:     if Neuer Prozess P* erreicht CPU then
6:         Q.enqueue(P*)
7:     end if
8: end while
```

Warteschlange genommen und abgearbeitet. Trifft ein neuer Prozess während der Abarbeitung eines anderen Prozesses ein, so wird dieser in die Warteschlange eingefügt.

Der große Vorteil von FCFS liegt in seiner Einfachheit und der daraus resultierenden leichten Implementierbarkeit. Darüber hinaus ist FCFS transparent und leicht vorhersehbar, da die Reihenfolge und Bearbeitungsdauer aller Prozesse nur von deren Ankunftszeit abhängt. Ein weiterer Vorteil liegt in der fairen Behandlung aller Prozesse, die ohne Bevorzugung erfolgt, da jeder Prozess in der Reihenfolge seines Eintreffens bearbeitet wird.

Allerdings hat FCFS auch erhebliche Nachteile, weshalb in der Praxis meist auf die alleinige Verwendung dieses Algorithmus verzichtet wird. Das Hauptproblem ist der Convoy-Effekt, bei dem ein langer Prozess, der früh in der Warteschlange erscheint, die nachfolgenden kürzeren Prozesse verzögert. Dies führt zu einer ineffizienten CPU-Auslastung und zu längeren Wartezeiten. Darüber hinaus berücksichtigt der FCFS neben der Dauer auch nicht die unterschiedliche Priorität von Prozessen, was insbesondere für interaktive Systeme nachteilig ist, in denen schnelle Antwortzeiten von höchster Relevanz sind [10, Kapitel 5].

Diese Mängel machen FCFS für viele moderne Anwendungen unbrauchbar. Daher wird im folgenden der OS Scheduling Algorithmus Round Robin näher betrachtet, der versucht, eine schnellere Antwortzeit zu ermöglichen.

2.2 Round Robin

Der Round Robin Scheduling Algorithmus ist ein weit verbreitetes OS-Scheduling-Verfahren, welches vor allem für seine Balance zwischen Fairness und Reaktionsfähigkeit bekannt ist. Es ist eines der ältesten aber immer noch sehr weit verbreitetsten Verfahren [11, S.158].

Beim Round-Robin-Algorithmus werden die anstehenden Prozesse wie zuvor beim FCFS-Prinzip zunächst in einer Warteschlange gesammelt. Zum Abarbeiten der Aufgaben wird jedem Prozess ein festes Zeitintervall, auch Zeit-Quantum oder Slice genannt, zugewiesen. Dieses Quantum ist in der Regel zwischen 10 und 100 Millisekunden lang und stellt die Dauer dar, welche die CPU nacheinander für einen Prozess aufbringt [12, S.209]. Der erste Prozess wird somit mit der Länge des Quantums bearbeitet und ist am Ende dieses entweder abgeschlossen oder muss unterbrochen werden. Der Algorithmus ist somit preemptive, da die Bearbeitung eines Prozesses unterbrochen werden kann. Ist dies der Fall, wird der Prozess an das Ende der Warteschlange gestellt und der nächste Prozess in der Warteschlange, ebenfalls mit dem gleichen Quantum, bearbeitet. Es entsteht somit ein zirkulares Verfahren, bei dem die Prozesse nacheinander in gleich großen Schritten bearbeitet werden. Dies gewährleistet, dass alle Prozesse regelmäßige CPU-Zeit erhalten und kein Prozess andere blockiert, wie es bei FCFS der Fall ist. Der Grobalgorithmus in 2 stellt vereinfacht diesen Ablauf des Round-Robin-Scheduling Verfahrens übersichtlich dar.

Algorithmus 2 Round Robin Scheduling

1: **Initialize:** Zeitquantum q, Prozesswarteschlange Q
2: **while** Prozesse existieren in Q **do**
3: Prozess $P \leftarrow Q$.dequeue()
4: Weise CPU P für Zeit $\min(P.\text{Restlaufzeit}, q)$ zu
5: **if** P.Restlaufzeit > 0 **then**
6: Q.enqueue(P) ▷ P ist nicht fertig, zurück in die Warteschlange
7: **end if**
8: **if** Q ist leer **then**
9: Warte auf neue Prozesse
10: **end if**
11: **end while**

Vor allem die Quantum-Länge spielt beim Round-Robin-Scheduling eine entscheidende Rolle, da der Wechsel zwischen den Prozessen, auch als Kontextwechsel bezeichnet, jedes Mal etwas Zeit beansprucht. Ein zu kurzes Quantum führt zu häufigen Kontextwechseln und erhöhtem Overhead, während ein zu langes Quantum die Reaktionszeiten verlängert, da Prozesse länger auf CPU-Zeit warten müssen. Ein angemessenes Quantum minimiert diesen Overhead und hält die Reaktionszeiten kurz. Ein Bereich von 20–50 Millisekunden für das Quantum ist oft ein guter Kompromiss, um die negativen Effekte von Kontextwechseln zu begrenzen und eine effiziente Prozessbearbeitung zu gewährleisten [11, S.158 f.].

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

Aufbauend 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 [12, S.214 f.].

Der Algorithmus in 3 stellt dar, wie das MLQ-Verfahren im Allgemeinen mit einer beliebigen Anzahl an Warteschlangen abläuft. In einem vereinfachen Beispiel mit zwei Warte-

Algorithmus 3 Multilevel-Queue-Scheduling

```
1: Initialize: n Warteschlangen Q_1, Q_2, ..., Q_n, jede mit ihrem eigenen Scheduling-
   Algorithmus S_1, S_2, ..., 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
```

schlangen 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.

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

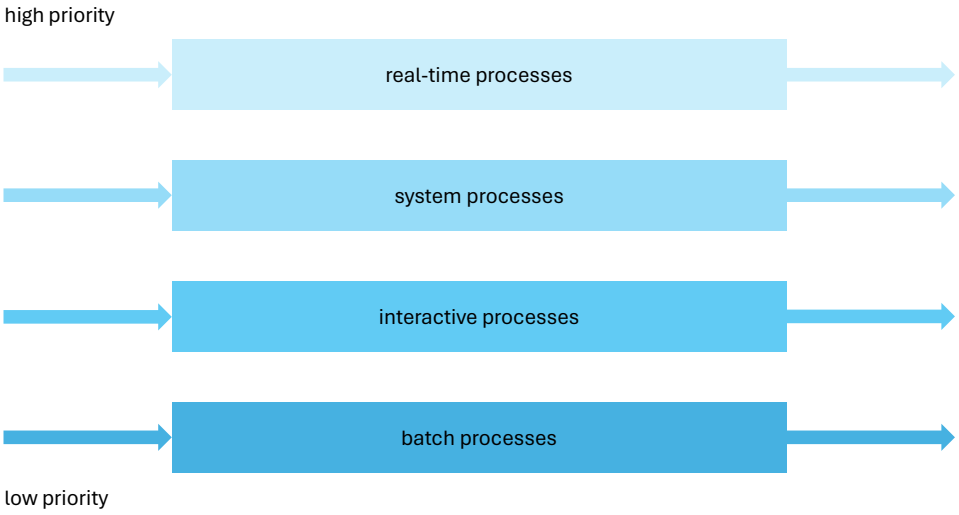

Abbildung 2.1: Darstellung des MLQ-Schedulings mit mehreren Warteschlangen [12, S.215]

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 eines Systems zu optimieren. Die zentrale Herausforderung hierbei ist die begrenzte Verfügbarkeit der CPU, da diese zu einem Zeitpunkt stets nur einen Prozess ausführen kann. Betriebssystem (OS) Scheduling Algorithmen versuchen den Einsatz der CPU zu optimieren, indem diese entscheiden welcher Prozess als nächstes bearbeitet werden soll. Die Auswahl des am besten geeigneten Algorithmus muss daher gewährleistet werden [13]. Dieses Kapitel beschäftigt sich daher mit den wichtigsten Metriken zur Auswahl der Algorithmen, 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 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 [14]. 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 [15]. 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 [15]. 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.

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

Durchschnittliche 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 [15]. In Formel 3.3 wird die Berechnung der Wartezeit dargestellt, welche die durchschnittliche Antwortzeit über alle n Prozesse ist.

$$\textit{Durchschnittliche Antwortzeit} = \frac{\sum_{i=1}^{n}(\textit{Startzeit}_i - \textit{Ankunftszeit}_i)}{n} \tag{3.3}$$

Durchschnittliche Wartezeit (Waiting Time): Die Wartezeit eines Prozesses ist die Gesamtzeit, welche dieser in der Warteschlange verbringt, bevor er Zugang zur CPU erhält [15]. 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.

$$\textit{Durchschnittliche Wartezeit} = \frac{\sum_{i=1}^{n}((\textit{Abschlusszeit}_i - \textit{Ankunftszeit}_i) - \textit{Bearbeitungszeit}_i)}{n} \tag{3.4}$$

Durchschnittliche Umlaufzeit (Turnaround Time): Bei der Umlaufzeit handelt es sich um die gesamte Dauer vom Start eines Prozesses bis zu dessen Abschluss [15]. 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.

$$\textit{Durchschnittliche Umlaufzeit} = \frac{\sum_{i=1}^{n}(\textit{Abschlusszeit}_i - \textit{Ankunftszeit}_i)}{n} \tag{3.5}$$

Fairness: Die Metrik der Fairness misst, wie gleichmäßig die CPU-Zeit zwischen den ver-

schiedenen Prozessen aufgeteilt wird. Ein fairer Scheduling-Algorithmus stellt sicher, dass kein Prozess übermäßig bevorzugt oder benachteiligt wird [16]. 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

Für eine Simulation der drei zuvor beschriebenen OS Scheduling Algorithmen ist die Erstellung eines abzuarbeitenden Datensatzes mit Prozessen essentiell. Mithilfe einer randomisierten Generierung der Prozesse wird versucht eine möglichst hohe Vielzahl von Szenarien abzudecken, um die Leistungsfähigkeit der Algorithmen unter verschiedenen Bedingungen zu testen. Jeder Prozess besitzt hierbei die eine Ankunftszeit, Bearbeitungsdauer und eine geringe oder hohe Priorisierung, wie zu sehen im Quelltext 3.1.

```
1 class Process:
2   def __init__(
3     self, id: int, arrival_time: int, burst_time: int, priority
          : str = "low"
4   ) -> None:
5         self.id = id
6         self.arrival_time = arrival_time
7         self.burst_time = burst_time
8         self.priority = priority
```

Quelltext 3.1: Prozess mit Attributen in Python implementiert

Für die randomisierte Generierung der Prozesseigenschaften wird zunächst die Ankunftszeit *arrival_time* der Prozesse bestimmt. Um eine realitätsnahe Simulation zu gewährleisten, wird die Ankunftszeit so variiert, dass sowohl gleichmäßig verteilte als auch in Clustern ankommende Prozesse im Datensatz vorhanden sind. Diese Variation wird durch Anpassung

der Ankunftszeit jedes nachfolgenden Prozesses erreicht, wobei eine zufällige Abweichung berücksichtigt wird, um die Clusterbildung zu simulieren.

Des Weiteren wird die Bearbeitungsdauer *burst_time* basierend auf einer Normalverteilung bestimmt, um realistische Variationen im Datensatz zu beinhalten. Hierbei sind die Parameter des Mittelwertes und der Standardabweichung zu wählen. Ein weiterer relevanter Parameter ist die Verteilung der Priorität *priority* innerhalb des Datensatzes. Da zu hohe oder geringe Werte zu einer Vernachlässigung der Vorteile von Round Robin und MLQ Scheduling führen, ist es wichtig hierbei eine Balance zu finden. Tabelle 3.1 zeigt die Wahl der unterschiedlichen Prozessparameter auf. Obwohl diese Parameter mit der Intention einer möglichst neutralen Verteilung der Prozesse sorgfältig festgelegt wurden, ist es wichtig zu betonen, dass die Prozessspezifikationen je nach Anwendungsfall abweichen können.

Parameter	Beschreibung
Anzahl der Prozesse	100 (für eine umfangreiche Evaluation)
Durchschnittliche Burst-Zeit	250 ms (durchschnittliche Ausführungszeit)
Standardabweichung der Burst-Zeit	600 ms (für breite Streuung der Anforderungen)
Prozentsatz der Hochprioritätsprozesse	20% (ein Fünftel aller Prozesse)
Variation der Ankunftszeit	100 ms (für unterschiedliche Ankunftsmuster)

Tabelle 3.1: Parameter für die Erstellung des Prozess-Datensatzes

Nach der Erstellung des Datensatzes werden diese Prozesse nun von FCFS, Round Robin und MLQ scheduled und abgearbeitet. Die zuvor erläuterten Metriken werden berechnet und in Tabelle 3.2 dargestellt.

Metrik	FCFS	Round Robin	MLQ
Durchschnittliche Wartezeit	43.65 ms	31.37 ms	43.02 ms
Durchschnittliche Umlaufzeit	339.73 ms	327.45 ms	339.10 ms
Unfairness-Index	148.68	91.22	147.14
Kontextwechsel	100.00	678.00	165.00

Tabelle 3.2: Vergleich der Scheduling-Algorithmen

Bei Betrachtung der Ergebnisse wird deutlich, dass Round Robin eine geringere durchschnittliche Wartezeit im Vergleich zu FCFS und MLQ aufweisen kann. Dies ist der Fall, da mithilfe der Einteilung in Quanten Prozesse gleichmäßiger verarbeitet und Bottlenecks vermieden werden. Obwohl MLQ ebenfalls diese Funktionsweise für die Prozesse mit hoher Priorität beinhaltet, sind diese Effekte nur marginal sichtbar. Ähnlich verhält es sich mit

der durchschnittlichen Umlaufzeit, welche neben der Wartezeit auch die Ausführungszeit beinhaltet. Auch beim Unfairness-Index, bei welchem geringe Werte zu priorisieren sind, da diese auf eine faire Abarbeitung der Prozesse hinweisen, wird deutlich, dass Round Robin die Prozesse gleichmäßiger abarbeiten kann. Diese Vorteile in der Wartezeit, Umlaufzeit und der Fairness haben allerdings den Preis einer höheren Anzahl an Kontextwechseln. Round Robin kann diese Vorteile nämlich nur erzielen, indem regelmäßig zwischen den Prozessen rotiert wird. Diese Kontextwechsel führen zu Overhead des gesamten Systems. Da die zeitliche Größe dieser Kontextwechsel sehr systemabhängig ist und von zahlreichen Faktoren abhängt, wurde diese nicht simuliert. Grundlegend können hierbei Werte zwischen wenigen Mikrosekunden bis hin zu etwa einer Mikrosekunde angenommen werden.

Die Simulation zeigt auf, dass FCFS, Round Robin und MLQ jeweils individuelle Stärken und Schwächen besitzen. Während FCFS die Anzahl der Kontextwechsel minimiert und hierdurch ein effizientes System darstellt, bieten Round Robin und MLQ deutliche Vorteile für interaktive Systeme, bei welchen eine schnelle Reaktion auf Prozesse hoher Priorität entscheidend ist. Die Auswahl der jeweiligen Scheduling Algorithmen kann daher nicht pauschalisiert werden, sondern hängt stark vom spezifischen Anwendungsfall ab.

4 Anwendungsgebiete

In diesem Kapitel wird auf konkrete Anwendungsgebiete der oben genannten OS-Scheduling-Algorithmen eingegangen. Da diese Algorithmen auch außerhalb von Betriebssystemen eine zentrale Rolle spielen, werden auch Beispiele für solche Anwendungsgebiete aufgenommen.

In modernen Betriebssystemen ist der Aufbau der Prozessverwaltung und die Anwendung von Scheduling-Algorithmen ein komplexes Thema, das in der Praxis eine Vielzahl von Anforderungen erfüllen muss. Die hier vorgestellten Algorithmen sind nur ein kleiner Ausschnitt aus der Vielzahl von Scheduling-Algorithmen, die in modernen Betriebssystemen zum Einsatz kommen.

Windows

Windows verwendet beispielsweise 7 Prioritätsstufen. Prozesse können sich selbst die folgenden Stufen zuweisen:

- IDLE
- BELOW NORMAL
- NORMAL
- ABOVE NORMAL
- HIGH
- REALTIME

Innerhalb eines Prozesses können die einzelnen Threads dann jeweils 7 Prioritätsebenen haben, die die Threads untereinander sortieren:

- IDLE
- LOWEST
- BELOW NORMAL
- NORMAL
- ABOVE NORMAL
- HIGHEST
- CRITICAL

Windows verwendet beide Prioritätszuweisungen, um dem Thread eine Basispriorität zwischen 0 und 31 zuzuweisen [17].

MacOS

Mit der Einführung der neuen M-Series Prozessoren unterteilt Apple die Prozessorkerne in Leistungs- und Effizienzkerne. Die Leistungskerne sind für rechenintensive Aufgaben zuständig, die Effizienzkerne für weniger rechenintensive Aufgaben. [18]

Im Allgemeinen verwendet MacOS, ähnlich wie Windows, verschiedene Prioritätsstufen oder Quality of Services, die in 4 Kategorien unterteilt werden können:

- Background
- Utility
- User Initiated
- User Interactive

Prozesse der Priorität Background werden nur auf den Effizienzkernen ausgeführt, während Prozesse von höherer Priorität auf beiden Prozessorkernarten ausgeführt werden können. Entwickler können das Verhalten eines Prozesses steuern und beispielsweise festlegen, dass ein User Initiated Process trotz des hohen Quality of Service nur den Effizienzkern verwenden soll. Laut hoakley wird innerhalb der einzelnen Prioritätsstufen eine Art FCFS verwendet, wobei die Prozesse in der Reihenfolge ihres Eintreffens abgearbeitet werden [19].

Linux

Da Linux ein Open-Source-Betriebssystem ist, kann hier genau angegeben werden, welches Scheduling-Verfahren verwendet wird. Im Jahr 2007 wurde das sogenannte Completely Fair Scheduler (CFS) eingeführt, welches alle Prozesse möglichst fair behandeln soll. Linux verwendet hier verschiedene Konzepte, um Fairness zu gewährleisten. Eine Besonderheit, die hier jedoch hervorsticht, ist die Verwendung eines Rot-Schwarz-Baumes, der die Prozesse in ihrer Ausführungsreihenfolge sortiert. Soll ein neuer Prozess hinzugefügt werden, kann dieser an der entsprechenden Stelle im Baum eingefügt werden. Der Scheduler wählt dann den Prozess aus, der sich auf dem äußersten linken Blatt des Baumes befindet und führt diesen aus. Der Vorteil dieses Baumes ist die Laufzeit beim Einfügen oder Entfernen eines Prozesses von $O(\log n)$, wobei n die Anzahl der Prozesse ist [20].

Im Jahr 2023 wurde der CFS durch den Earliest Eligible Virtual Deadline First (EEVDF) ersetzt. Dieser ergänzt den CFS um einige Punkte. So wird die Zeit, die ein Prozess auf der CPU verbringt; mit der Zeit, die er auf der CPU verbringen sollte; verglichen. Dieser Wert und eine zusätzlich vom Prozess angegebenen Dringlichkeit werden dann bei

der erneuten Einsortierung des Prozesses in die „Warteschlange“ berücksichtigt. Dadurch können dringliche Prozesse sowie Prozesse die weniger Zeit in der CPU verbracht haben bevorzugt werden [21].

Außerhalb von Betriebssystemen

FCFS wird in zahlreichen Gebieten verwendet. Groover erklärt beispielsweise den Einsatz beim Planen von Produktionsabfolgen. Hier wird unter anderem FCFS verwendet um von dessen Fairness zu profitieren [22, S.735].

Ein bekanntes Anwendungsgebiet von Round Robin ist die Telekommunikation. Hierbei wird Round Robin für die Paketvermittlung und Lastverteilung in Netzwerkroutern eingesetzt [23, Kapitel 4.2.4].

A Ressourcen

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

Das Video kann auf Youtube unter folgendem Link gefunden werden: https://youtu.be/9bVAGvbL8Gc

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] Kurt Koffka. *Principles of Gestalt Psychology*. en. Google-Books-ID: cLnqI3dvi4kC. Psychology Press, 1999. ISBN: 978-0-415-20962-5.

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

[6] 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).

[7] John A. Sloboda. *The Musical Mind: The Cognitive Psychology of Music*. en. Oxford University Press, Apr. 1986. ISBN: 978-0-19-170625-7. DOI: 10.1093/acprof:oso/9780198521280.001.0001. URL: https://academic.oup.com/book/5183 (besucht am 23. 02. 2024).

[8] Roger C. Schank. *Tell me a story: A new look at real and artificial memory*. Tell me a story: A new look at real and artificial memory. Pages: xiii, 253. New York, NY, England: Charles Scribner'S Sons, 1990. ISBN: 978-0-684-19049-5.

[9] Richard John Anthony. „Chapter 2 - The Process View“. In: *Systems Programming*. Hrsg. von Richard John Anthony. Boston: Morgan Kaufmann, 2016, S. 21–106. ISBN: 978-0-12-800729-7. DOI: https://doi.org/10.1016/B978-0-12-800729-7.00002-9. URL: https://www.sciencedirect.com/science/article/pii/B9780128007297000029.

[10] Peter Baer Galvin, Silberschatz, Abraham, Gagne und Greg. *Operating system concepts; Seventh Edition*. John Wiley & Sons, 2004.

[11] Andrew S. Tanenbaum und Herbert Bos. *Modern operating systems*. Fifth edition, global edition. Hoboken, New Jersey: Pearson, 2024. ISBN: 9781292727899. DOI: Andrew. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=7222243.

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

[13] 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).

[14] 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).

[15] 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).

[16] 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).

[17] Karl-Bridge-Microsoft et al. *Planungsprioritäten - Win32 apps*. 2023. URL: https://learn.microsoft.com/de-de/windows/win32/procthread/scheduling-priorities.

[18] hoakley. *Scheduling of Processes on M1 Series Chips: first draft*. 2022. URL: https://eclecticlight.co/2022/01/13/scheduling-of-processes-on-m1-series-chips-first-draft/.

[19] hoakley. *Scheduling of Threads on M1 Series Chips: second draft*. 2022. URL: https://eclecticlight.co/2022/01/25/scheduling-of-threads-on-m1-series-chips-second-draft/.

[20] M. Jones. *Inside the Linux 2.6 Completely Fair Scheduler*. Hrsg. von IBM Developer. 2009. URL: https://developer.ibm.com/tutorials/l-completely-fair-scheduler/.

[21] Oliver Müller. „Linux 6.6 bringt neuen Scheduler“. In: *heise online* (1/11/2023). URL: https://www.heise.de/news/Linux-6-6-bringt-neuen-Scheduler-9350044.html.

[22] Mikell P. Groover. *Automation, production systems, and computer-integrated manufacturing*. Pearson Education India, 2016.

[23] James Kurose und Keith Ross. *Computer networks: A top down approach featuring the internet*. Pearson, 2010.