19.Vorlesung

Betriebssysteme (BTS)

Christian Baun
cray@unix-ag.uni-kl.de

Hochschule Mannheim – Fakultät für Informatik
Institut für Betriebssysteme

22.6.2007

Heute

- 2.Testklausur mit 22 Fragen aus den Vorlesungen 9 bis 17.

Aufgabe 1

Was ist ein **Dispatcher** und was sind seine Aufgaben?

- Beim Dispatching wird der Prozesswechsel durchgeführt.
- Aufgabe des Dispatchers (Prozessumschalters) ist es, die Zustandsübergänge der Prozesse durchzuführen.
- Bei einem Prozesswechsel entzieht der Dispatcher dem derzeit aktiven, rechnenden Prozess die CPU und teilt sie dem Prozess zu, der in der Warteschlange an erster Stelle steht.
- Bei Übergängen zwischen den Zuständen `bereit` und `blockiert` werden vom Dispatcher die entsprechenden Prozesskontrollblöcke aus den Zustandlisten entfernt und entsprechend neu eingefügt.
- Übergänge aus oder in den Zustand `rechnend` bedeuten immer einen Wechsel des aktuell rechnenden Prozesses auf der CPU.

Aufgabe 2

Was ist ein **Scheduler** und was sind seine Aufgaben?

- Der Scheduler eines Betriebssystems legt Ausführungsreihenfolge der Prozesse im Zustand `bereit` fest.
- Die Entscheidung, welcher Prozess wann an der Reihe ist, ist vom Scheduling-Algorithmus abhängig.
- Beim Scheduling wird versucht folgende Grundsätze einzuhalten:
  - **Durchsatz**: Abarbeitung möglichst vieler Prozesse pro Zeitintervall.
  - **Effizienz**: Möglichst vollständige Auslastung der CPU.
  - **Termineinhaltung**: Prozesse, die zu einem bestimmten Termin abgearbeitet werden müssen, werden so eingeplant, dass der Termin eingehalten wird.
  - **Fairness**: Die CPU den Prozessen möglichst gerecht zuteilen.
  - **Overhead**: Der zeitliche Aufwand für das Scheduling soll minimal sein.
  - **Turnaround**: Die Wartezeit der Benutzer soll minimal sein.

Aufgabe 3

Die existierenden **Schedulingverfahren** können in zwei grundsätzliche Klassen unterteilt werden. Welche sind das und in was unterscheiden sich diese?

- **Nicht-präemptives Scheduling** (nicht-verdrängendes Scheduling): Ein Prozess, der vom Scheduler die CPU zugewiesen bekommen hat, behält die Kontrolle über diese bis zu seiner vollständigen Fertigstellung. Eine vorzeitige Entziehung der CPU durch den Scheduler ist nicht vorgesehen. Problematisch ist dabei, dass ein Prozess die CPU so lange belegen kann, wie er möchte und andere, vielleicht dringendere Prozesse für lange Zeit nicht zum Zuge kommen.
- **Präemptives Scheduling** (verdrängendes Scheduling): Einem Prozess kann die CPU vor seiner Fertigstellung wieder entzogen werden, um diese anderen Prozessen zuzuteilen. Der Prozess pausiert so lange in seinem aktuellen Zustand, bis ihm wieder die CPU vom Scheduler zugeteilt wird.

Aufgabe 4

Nennen Sie vier unterschiedliche **Scheduling-Verfahren** (Algorithmen).

- **Round Robin** (RR) mit Zeitquantum
- **First Come First Served** (FCFS) bzw. **First In First Out** (FIFO)
- **Shortest Job First** (SJF) und **Longest Job First** (LJF)
- **Shortest Remaining Time First** (SRTF)
- **Longest Remaining Time First** (LRTF)
- **Prioritätengesteuertes Scheduling**
- **Earliest Deadline First** (EDF)
- **Fair-Share**
- **Multilevel-Scheduling**

Aufgabe 5:

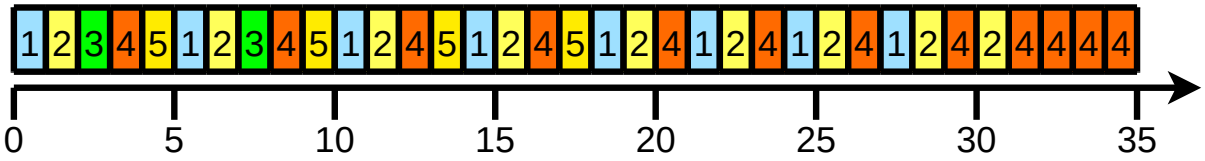
Auf einem Einprozessorrechner sollen fünf Prozesse verarbeitet werden:

Prozess	CPU-Laufzeit (ms)	Priorität
1	8	12
2	9	3
3	2	10
4	12	4
5	4	6

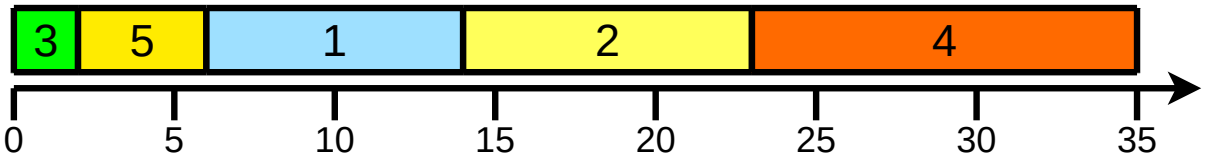
Hohe Prioritäten sind durch hohe Zahlen gekennzeichnet. Skizzieren Sie die Ausführungsreihenfolge der Prozesse mit einem Gantt-Diagramm (Zeitleiste) für **Round Robin** (Zeitquantum $q = 1$ ms), **SJF** und **Prioritätengesteuertes Scheduling**. Berechnen Sie die mittleren Laufzeiten und Wartezeiten der Prozesse.

Aufgabe 5 (Fortsetzung)

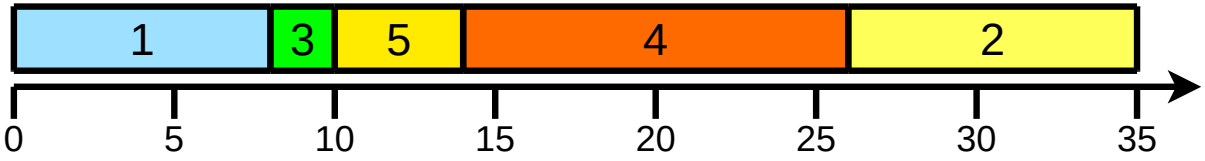
Round Robin
(Zeitquantum = 1)



Shortest Job First



Prioritätengesteuertes
Scheduling



Aufgabe 5 (Fortsetzung)

- Laufzeit (Turnaround Time) der Prozesse

	P1	P2	P3	P4	P5
Round Robin	28	31	8	35	18
Shortest Job First	14	23	2	35	6
Prioritätengesteuertes Scheduling	8	35	10	26	14

Round Robin	$\frac{28+31+8+35+18}{5}$	$=$	24 ms
Shortest Job First	$\frac{14+23+2+35+6}{5}$	$=$	16 ms
Prior. Scheduling	$\frac{8+35+10+26+14}{5}$	$=$	$18,6$ ms

Aufgabe 5 (Fortsetzung)

- Wartezeit der Prozesse – Zeit in der `bereit`-Liste

	P1	P2	P3	P4	P5
Round Robin	20	22	6	23	14
Shortest Job First	6	14	0	23	2
Prioritätengesteuertes Scheduling	0	26	8	14	10

Round Robin	$\frac{20+22+6+23+14}{5}$	$=$	17 ms
Shortest Job First	$\frac{6+14+0+23+2}{5}$	$=$	9 ms
Prior. Scheduling	$\frac{0+26+8+14+10}{5}$	$=$	$11,6$ ms

Aufgabe 6:

Was ist **kooperatives Scheduling**? Was sind die Vor- und Nachteile von kooperativem Scheduling?

- Beim kooperativen Scheduling besteht die Möglichkeit, dass Prozesse die CPU freiwillig an andere Prozesse abgeben.
- Die freiwillige Prozessoraufgabe kann immer dann zum Einsatz kommen, wenn ein Prozess auf ein Ereignis wartet oder ein Ereignis sehr häufig auftritt. Beispiele sind das Warten auf die Ergebnisse anderer Prozesse, Tastatureingaben und das Laden von Dateien über Netzwerke.

Aufgabe 6 (Fortsetzung)

- Das kooperativen Schedulings hat zwei grundlegende Nachteile:
  1. Die freiwillige Prozessoraufgabe wird nicht vom Betriebssystem initiiert. Wenn ein Prozess hängenbleibt (abstürzt), bevor er die Kontrolle an einen anderen Prozess abgegeben hat, kann es dazu kommen, dass das ganze Betriebssystem nicht mehr reagiert (hängt).
  2. Damit das kooperative Scheduling funktioniert, ist eine saubere Anwendungsentwicklung notwendig. Die freiwillige Prozessoraufgabe muss an den geeigneten Stellen, hauptsächlich in Schleifen, in denen auf das Eintreffen von Ereignissen gewartet wird, in den Programmen implementiert sein.

Aufgabe 7:

Was versteht man im Bereich der Interprozesskommunikation unter **kritischen Abschnitten** und was muss bei kritischen Abschnitten beachtet werden?

- Auf allen Systemen, auf denen mehrere Prozesse laufen, müssen die Prozesse Daten austauschen und Ressourcen gemeinsam nutzen.
- In kritischen Abschnitten greifen die Prozesse nicht nur lesend auf gemeinsame Daten zu.
- Kritische Abschnitte dürfen nicht von mehreren Prozessen gleichzeitig durchlaufen werden.
- Damit Prozesse auf gemeinsam genutzten Speicher zugreifen können, brauchen wir **wechselseiteigen Ausschluss** (Mutual Exclusion). Das bedeutet, dass ein Prozess auf eine gemeinsam genutzte Datei oder Variable zugreifen kann, ohne dass ein anderer Prozess ebenfalls zugreifen kann.

Aufgabe 8:

Was ist eine **Race Condition** und wie können Race Conditions verhindert werden?

- Eine **Race Condition** (*Wettlaufsituation*) bezeichnet eine Konstellation, bei der das Ergebnis eines Prozesses von der Reihenfolge oder dem zeitlichen Ablauf anderer Ereignisse abhängt.
- Bei Race Conditions kommt es zu einer unbeabsichtigten Wettlaufsituation zweier Prozesse, die auf die gleiche Speicherstelle zugreifen wollen, um deren Wert zu verändern.
- Race Conditions sind ein häufiger Grund für schwer auffindbare Programmfehler.
- Eine Möglichkeit, Race Conditions zu vermeiden ist der Einsatz von Semaphoren.

Aufgabe 9:

Welche beiden Probleme können beim Einsatz von **Sperren** entstehen? Nennen Sie die beiden möglichen Probleme und erklären Sie diese mit jeweils in wenigen Sätzen.

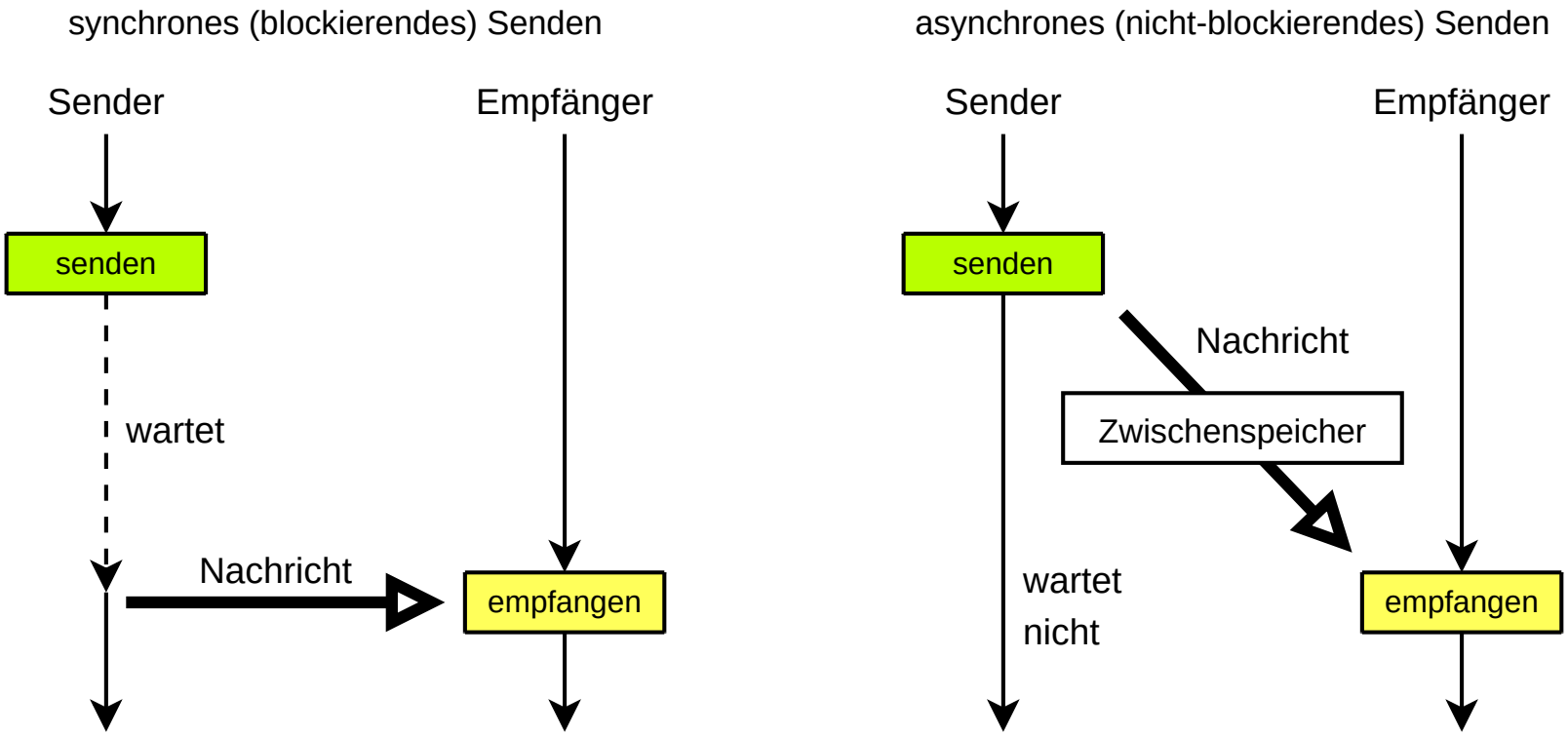
- **Verhungern** (Starving)
  - Hebt ein Prozess eine Sperre nicht wieder auf, müssen die anderen Prozesse unendlich lange auf die Freigabe warten. In einem solchen Fall spricht man vom Verhungern der Prozesse.
- **Deadlock**
  - Warten zwei Prozesse gegenseitig auf die von ihnen gesperrten Ressourcen, sperren sich die Prozesse gegenseitig. Es kommt zu einer Verklemmung (Deadlock).
  - Da alle am Deadlock beteiligten Prozesse warten, kann keiner ein Ereignis auslösen, so dass ein anderer geweckt wird. Beim Deadlock warten alle beteiligten Prozesse ewig.

Aufgabe 10:

Erklären Sie in wenigen Sätzen die Unterschiede zwischen **synchronem** und **asynchronem Senden**. Verdeutlichen Sie die Unterschiede mit je einem Diagramm.

- Beim **synchronen**, dem **blockierenden Senden**, ist der Sender so lange in der Send-Funktion blockiert, bis der Empfänger empfangsbereit ist. Erst wenn der Empfänger empfangsbereit ist, findet die Übertragung der Nachricht statt.
- Beim **asynchronen**, dem **nicht-blockierenden Senden**, wird die Nachricht des Senders in einem Zwischenspeicher (Puffer) abgelegt. Der Empfänger kann die Nachricht sofort oder später aus dem Zwischenspeicher abholen. Der Sender fährt nach der Ablage der Nachricht im Zwischenspeicher sofort mit seiner Ausführung fort.

Aufgabe 10 (Fortsetzung)

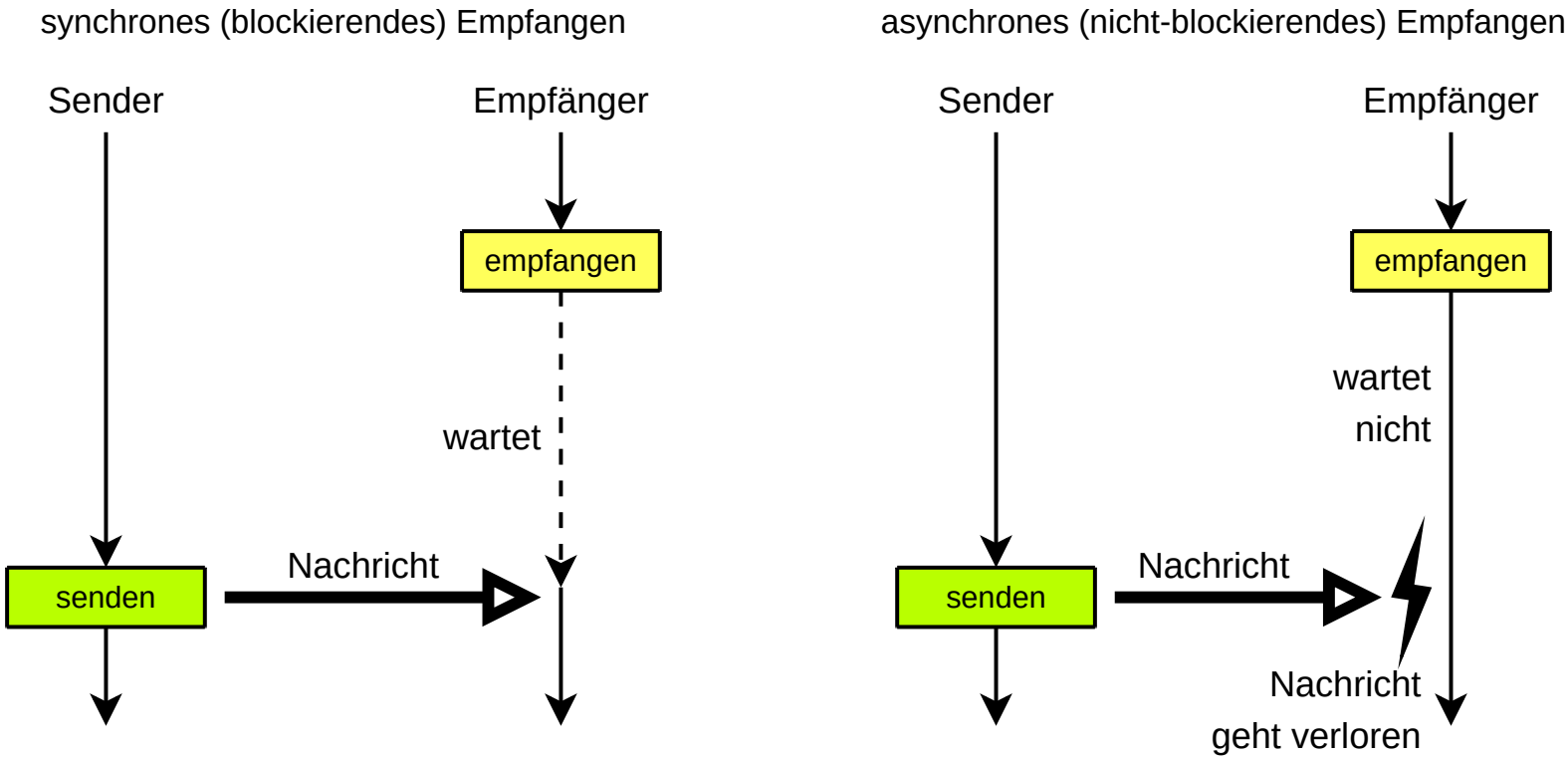


Aufgabe 11:

Erklären Sie in wenigen Sätzen die Unterschiede zwischen **synchronem** und **asynchronem Empfangen**. Verdeutlichen Sie die Unterschiede mit je einem Diagramm.

- Beim **synchronen**, dem **blockierenden Empfangen**, ist der Empfänger so lange blockiert, bis er die Nachricht des Senders erhält.
- Beim **asynchronen**, dem **nicht-blockierenden Empfangen**, erhält der Empfänger die Nachricht nur, wenn sie bereits vorliegt. Liegt die Nachricht nicht rechtzeitig vor, verzichtet der Empfänger auf die Nachricht und setzt seine Ausführung fort. Die Nachricht geht verloren.

Aufgabe 11 (Fortsetzung)



Aufgabe 12:

Welches sind die drei möglichen Zugriffsoperationen auf **Semaphoren**? Erklären Sie in wenigen Sätzen deren Aufgabe und Funktion.

- **Initialisierung**: Zuerst wird ein Semaphor erzeugt oder, ein bestehendes Semaphor geöffnet. Bei einem neuen Semaphor wird zu Beginn die Zählvariable mit einem nichtnegativen Anfangswert initialisiert.
- **P-Operation** (*passieren*): Diese Operation prüft zu allererst, ob der Wert der Zählvariable gleich 0 ist. Ist der Wert 0, wird der Prozess blockiert. Ist der Wert größer 0, wird er um 1 erniedrigt.
- **V-Operation** (*freigeben*): Diese Operation erhöht als erstes den Wert der Zählvariable um 1. Befinden sich Prozesse im Warteraum, wird ein Prozess deblockiert. Der gerade deblockierte Prozess setzt dann seine P-Operation fort und erniedrigt als erstes die Zählvariable.

Aufgabe 13:

Was ist ein **Mutex** und was sind die Unterschiede zwischen Mutexen und Semaphoren?

- Mutexe dienen dem Schutz kritischer Abschnitte, auf denen zu jedem Zeitpunkt immer nur **ein Prozess** zugreifen darf.
- Mutexe sind effizienter und einfacher zu realisieren als Semaphore, da sie nur zwei Zustände annehmen können, nämlich **belegt** und **nicht belegt**.
- Folglich wird nur ein Bit benötigt, um die Information darzustellen. In der Realität wird häuig eine Ganzzahl verwendet, bei der 0 **nicht belegt** und jeder andere Wert **belegt** bedeutet.
- Wenn die Möglichkeit eines Semaphors zu zählen nicht benötigt wird, kann man die vereinfachte Version eines Semaphors, Mutexe, vorziehen.

Aufgabe 14:

Drei Schwimmer sollen hintereinander eine bestimmte Strecke schwimmen. Der erste Schwimmer soll als erstes starten.

Der zweite Schwimmer darf erst starten, wenn der erste Schwimmer im Ziel angekommen ist.

Der dritte Schwimmer darf erst starten, wenn der zweite Schwimmer im Ziel angekommen ist.

Entwerfen Sie eine korrekte Lösung in Pseudocode.

```
// Initialisierung der Semaphoren
s_init (Sema1, 0);
s_init (Sema2, 0);

task Erster is
        < schwimmen >
        V(Sema1);

task Zweiter is
        P(Sema1);
        < schwimmen >
        V(Sema2);

task Dritter is
        P(Sema2);
        < schwimmen >
```

Aufgabe 15:

Beschreiben Sie eine sicher funktionierende Lösungsmöglichkeit für das **Problem der speisenden Philosophen**.

- Eine Lösung, bei der immer zwei Philosophen essen können und weder Deadlocks noch Lifelocks entstehen können, ist wie folgt:
  - In einem Feld wird verfolgt, ob ein Philosoph gerade nachdenkt, isst oder hungrig ist, also versucht die Gabeln zu bekommen.
  - Ein Philosoph kann nur in den Zustand essen übergehen, wenn seine beiden Nachbarn gerade nicht am Essen sind.
  - Es existiert pro Philosoph ein Semaphor, so dass hungrige Philosophen blockieren können, falls die benötigten Gabeln in Gebrauch sind.
  - Erst gelangt ein hungriger Philosoph alleine in den kritischen Bereich. Er prüft, ob seine beiden Nachbarn essen. Falls ja, blockiert er solange. Falls nein, beginnt er zu essen und verlässt den kritischen Bereich.
  - Ist er fertig mit Essen, muss er in den kritischen Bereich kommen, seinen Status auf *nachdenken* setzen, die Gabeln freigeben und den kritischen Bereich verlassen.

Aufgabe 16:

Beschreiben Sie die Unterschiede zwischen **relativen** und **absoluten Pfadangaben**. Was sind die Vor- und Nachteile?

- Hat eine Datei einen **absoluten Pfadnamen**, beschreibt dieser den kompletten Pfad von der Wurzel bis zur Datei. Ein absoluter Pfad wird immer funktionieren, egal wie das aktuelle Verzeichnis ist. Ein Beispiel für einen absoluten Pfadnamen ist:
  `/usr/src/linux/arch/i386/boot/bzImage`
- Ein **relativer Pfadname** wird immer in Verbindung mit dem aktuellen Verzeichnis gesehen. Alle Pfade, die nicht mit dem Wurzelverzeichnis beginnen, sind relative Pfade. Relative Pfadnamen sind meistens kürzer als absolute Pfadnamen. Ein Beispiel für einen relativen Pfadnamen ist:
  `BTS_SS2007/Vorlesung_15/folien_bts_vorlesung_15.tex`

Aufgabe 17:

Welche Auswirkungen hat die Größe der **Cluster** im Dateisystem?

- Je kleiner die Cluster, desto größer der Overhead entsteht.
- Je größer die Cluster, desto mehr Speicher geht durch interne Fragmentierung verloren.
- Ein Beispiel:
  - Alle zu schreibenden Dateien haben eine Dateigröße von 2 Kilobyte.
  - Ist die Clustergröße 2 Kilobyte, liegt keine interne Fragmentierung vor und es geht kein Speicherplatz verloren.
  - Ist die Clustergröße 4 Kilobyte, gehen gehen pro Cluster 2 Kilobyte verloren, also die Hälfte des vorhandenen Speicherplatzes.
  - Ist die Clustergröße 8 Kilobyte, gehen pro Cluster 6 Kilobyte verloren. Es können also nur 25 Prozent des vorhandenen Speicherplatzes genutzt werden.

Aufgabe 18:

Was sind **Journaling-Dateisysteme**? Wie ist die Funktionsweise von Journaling-Dateisystemen? Was sind die Vorteile von Journaling-Dateisystemen gegenüber Dateisystemen ohne Journal?

- Ein Journaling-Dateisystem führt ein sogenanntes Journal über die Daten, auf die Schreibzugriffe durchgeführt werden sollen.
- Eine zu ändernde Datei behält ihre Gültigkeit, bis die Schreibzugriffe durchgeführt wurden.
- In festen Zeitabständen wird das Journal geschlossen und die Schreiboperationen werden durchgeführt.
- Während der Abarbeitung des Journals wird festgehalten, welche Schreiboperationen bereits erfolgreich durchgeführt wurden.
- Gleichzeitig werden Prüfsummen von Daten von der Änderung erstellt.

Aufgabe 18 (Fortsetzung)

- Mit Hilfe des Journals und den Prüfsummen können nach einem Systemabsturz die zu überprüfenden bzw. wiederherzustellenden Dateien schnell identifiziert und repariert werden und das Journal wenn möglich weiter abgearbeitet werden.
- Im schlimmsten Fall gehen Änderungsanforderungen, die im Journal vermerkt waren, verloren. Die Dateien auf dem Medium bleiben aber in einem konsistenten Zustand.
- Das führen eines Journals führt zu geringen Leistungseinbußen.
- Der Vorteil von Journaling-Dateisystemen ist, dass bei einem Absturz des Betriebssystems nicht alle Daten überprüft, sondern nur die zu dem Zeitpunkt geöffneten Daten repariert werden müssen. Das führt zu großen Zeitersparnissen.

Aufgabe 19:

Was versteht man unter **Defragmentierung**? Macht es Sinn unter Betriebssystemen mit Mehrprogrammbetrieb zu defragmentieren? Begründen Sie Ihre Aussage.

- Durch das Schreiben von Daten auf einen Datenträger kommt es zwangsläufig immer zu Fragementierung, also die Daten sind nicht mehr zusammenhängend angeordnet. Bei der Defragmentierung sollen zusammengehörige Daten wieder räumlich zusammenhängend auf dem Datenträger angeordnet werden.
- Nur wenn die Suchzeiten bei der Gesamtwartezeit auf die Daten sehr groß sind, macht eine Defragmentierung Sinn. Das ist aber nur bei Betriebssystemen mit Einzelprogrammbetrieb (Singletasking), die kaum Hauptspeicher zum Cachen der Festplattenzugriffe verwenden der Fall.

Aufgabe 19 (Fortsetzung)

- Bei Betriebssystemen mit Mehrprogrammbetrieb (Multitasking) laufen immer mehrere Programme und es steht ausreichend Cache im RAM zur Verfügung.
- Es werden fast nie Anwendungen große Datenmengen am Stück lesen wollen oder auch können, ohne dass andere Anwendungen weitere Zugriffe an anderen Stellen des Mediums dazwischenschieben.
- Die Wirkung des Cache überwiegt bei weitem die kurzzeitigen Vorteile, die eine Defragmentierung hätte.
- Defragmentierung hat mehr einen Benchmark-Effekt.
- In der Realität bringt Defragmentierung fast nichts.

Aufgabe 20:

Was versteht man unter **Virtualisierung** und was sind die Unterschiede zwischen Virtualisierung und **Hardware-Emulation**?

- Bei der **Virtualisierung** werden die Ressourcen eines Rechnersystems aufgeteilt und können von mehreren unabhängigen Betriebssystem-Instanzen genutzt werden.
- Es gibt in der Virtualisierung mehrere grundsätzlich verschiedene Konzepte und Technologien, die den Begriff Virtualisierung verwenden.
- Eine virtuelle Maschine ist ein nachgebildeter Rechner, der in einer abgeschotteten Umgebung auf einer realen Maschine läuft.
- Jede virtuelle Maschine verhält sich wie ein vollwertiger Computer mit eigenen Komponenten, wie CPU, Hauptspeicher, Festplatten, Grafikkarte, Netzwerkkarten, usw.

Aufgabe 20 (Fortsetzung)

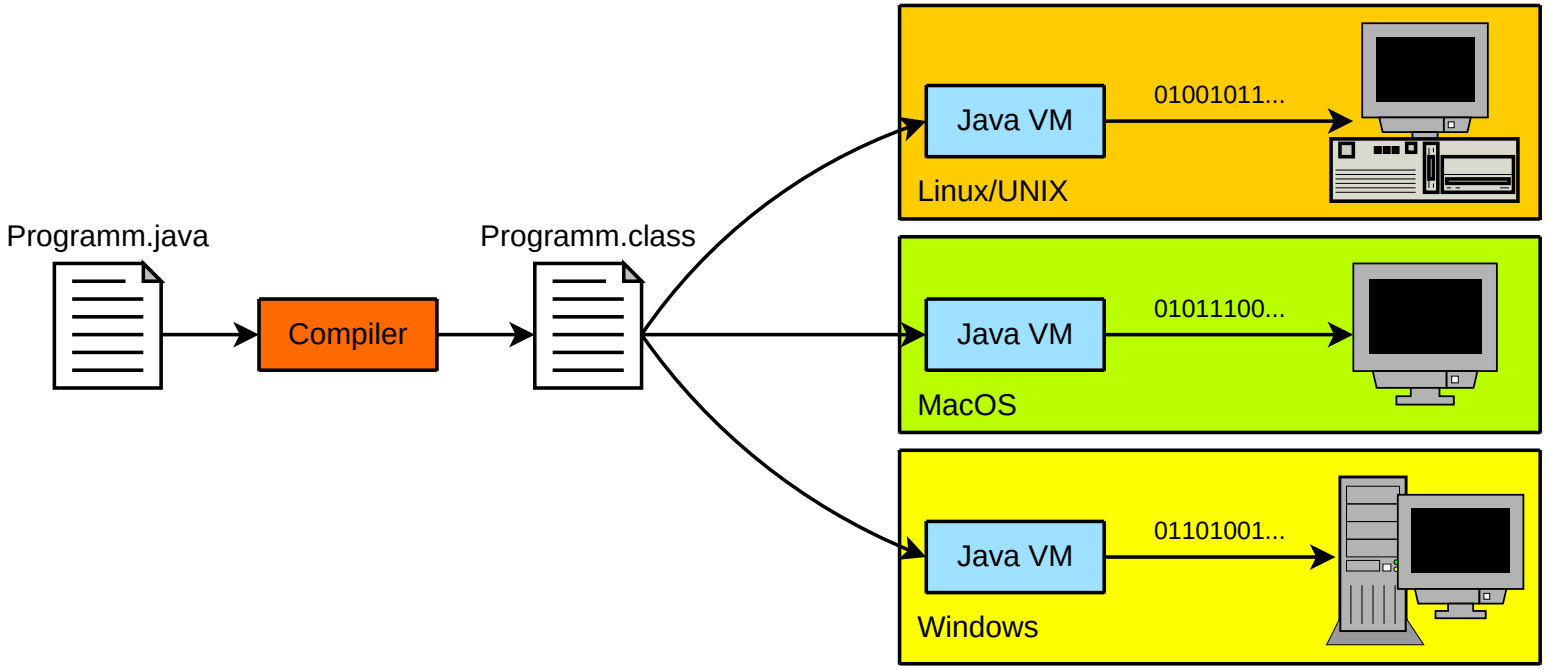
- Auf einige der Hardwarekomponenten des Computers kann eine virtuelle Maschine direkt zugreifen. Beispiele sind hier die CPU und der Hauptspeicher.
- In einer virtuellen Maschine kann ein Betriebssystem mit Applikationen genau wie auf einem realen Computer installiert werden. die Software merkt nicht, dass Sie sich in Wirklichkeit in einer virtuellen Maschine befindet
- Anforderungen des Gast-Betriebssystems werden unbemerkt von der Virtualisierungssoftware abgefangen und auf die real vorhandene oder emulierte Hardware umgesetzt.
- Bei der **Emulation** wird in den meisten Fällen versucht, die komplette Hardware eines Rechnersystems nachzubilden und so einem unveränderten Betriebssystem, das für eine andere Hardwarearchitektur (CPU) ausgelegt ist, den Betrieb zu ermöglichen.

Aufgabe 21:

Beschreiben Sie das Virtualisierungskonzept der **Applikationsvirtualisierung** am Beispiel der Java Virtual Machine.

- Bei der Applikationsvirtualisierung werden Anwendungen lokal, unter Verwendung lokaler Ressourcen in einer virtuellen Umgebung ausgeführt, die alle Komponenten bereitstellt, die die Anwendung benötigt.
- Die Java Virtual Machine (JVM) ist der Teil der Java-Laufzeitumgebung (JRE), der für die Ausführung des Java-Bytecodes verantwortlich ist. Die JVM ist für die Java-Programme die Schnittstelle zum Rechnersystem und dessen Betriebssystem.
- Der große Vorteil der Applikationsvirtualisierung ist Plattformunabhängigkeit. Nachteilig ist die geringere Ausführungsgeschwindigkeit gegenüber nativer Programmausführung.

Aufgabe 21 (Fortsetzung)



- Der Java-Compiler `javac` übersetzt den Quellcode in architekturunabhängige `.class`-Dateien, die Bytecode enthalten, der in der Java Virtual Machine lauffähig ist. Das `java`-Programm startet eine Java-Applikation in einer Instanz der Java Virtual Machine.

Aufgabe 22:

Beschreiben Sie das Konzept der **Paravirtualisierung** und die Vor- und Nachteile dieses Virtualisierungskonzepts.

- Bei der Paravirtualisierung wird keine Hardware virtualisiert oder emuliert.
- Virtuell gestartete Betriebssysteme verwenden eine abstrakte Verwaltungsschicht um auf die physischen Ressourcen wie Speicherplatz, Ein-/Ausgabegeräte und Netzwerkinterfaces zuzugreifen. Den Gast-Betriebssystemen steht keine emulierte Hardwareebene zur Verfügung, sondern eine API.
- Vorteil: Keine Geschwindigkeitseinbußen wie beim Konzept des Virtual Machine Monitor.
- Nachteil: Die Kernel der Gast-Betriebssysteme müssen speziell für den Betrieb in einem paravirtualisierten Kontext angepasst sein.

Nächste Vorlesung:

22.6.2007