

Betriebssysteme

Tutorium 6

Philipp Kirchhofer

`philipp.kirchhofer@student.kit.edu`

`http://www.stud.uni-karlsruhe.de/~uxbtt/`

**Lehrstuhl Systemarchitektur
Universität Karlsruhe (TH)**

28. Oktober 2009

- Studium der Informatik (Diplom)
- 7. Semester
- Vertiefungsgebiete
 - Betriebssysteme
 - Rechnerarchitektur
 - Eingebettete Systeme
 - Mikro- und Nanoelektronik
- Bei Fragen: philipp.kirchhofer@student.kit.edu

Was machen wir heute?

1 Organisatorisches

- Theorieblätter
- Programmieraufgaben
- Schein

2 Tutorien Übungsblatt

- Linux Basics
- Bits & Bytes
- OS Basics
- Hardware Basics

Tutorien

- Wöchentliches Tut-Blatt
 - Vorbereitung auf Theorieblätter
- Rückgabe der korrigierten Theorieblätter
- Fragen & Antworten zum Stoff
- Folien sind unter <http://www.stud.uni-karlsruhe.de/~uxbtt/> verfügbar

Theorieblätter

- Erscheinen 14-tägig
- Bearbeitung in Gruppen möglich, aber einzelne Abgabe
- Abgabe im „Briefkasten“ am IBDS (Infobau, Erster Stock, Nordflügel)
- Rückgabe der korrigierten Blätter im Tutorium

Programmieraufgaben

- 3 Stück
 - Synchronisation und Deadlocks
 - Speicherverwaltung
 - Systemaufrufe
- Erweiterung von OS/161 (MIPS basiertes Betriebssystem)
- Literaturhinweise:
 - **C von A bis Z, Galileo Computing** (Einführung in C)
 - **MIPS R2000 Befehlssatz**
 - **Programming Assignment 0**
- Abgabe in Zweiergruppen, Anmeldung bis 18. November per Mail an Tutor
 - Name und Matrikelnummer der Gruppenteilnehmer, Gruppenname
- Lösung (kompletter Sourcecode, kein Diff, komprimiert) vor Ablauf der jeweiligen Deadline an philipp.kirchhofer@student.kit.edu und Kopie (CC) an os161@ira.uka.de
- Korrektur: Funktionalität & Demonstration

Punkte Bachelor

- Schein nötig zum Bestehen des Moduls
- Mindestens 100 Punkte für Schein notwendig
- Insgesamt 190 Punkte erreichbar
 - 120 aus Theorieblättern
 - 70 aus Programmieraufgaben
- Bonusregelung:
 - Mindestens 110 Punkte – 1 Bonuspunkt
 - Mindestens 130 Punkte – 2 Bonuspunkte
 - Mindestens 150 Punkte – 3 Bonuspunkte
 - Mindestens 170 Punkte – 4 Bonuspunkte

Punkte Diplom

- Kein Schein nötig
- Klausurbonus wie bei Bachelor
- Punkte vom Vorjahr: Maximum aus Gesamtbonuspunkten von früheren und aktuellen Gesamtbonuspunkten

Frage 1.1.a

Was ist eine Shell?

Antwort

Eine Shell ist ein text-basiertes Programm, mit dessen Hilfe der Benutzer mit dem Betriebssystem interagieren kann. Die grundlegenden Funktionalitäten einer Shell beinhalten Funktionen zum Anzeigen und Verändern des Dateisystems sowie der Aufruf von weiteren Anwendungen.

Linux Standard-Befehle

Frage 1.1.b

Wie kann man unter Linux einfach Hilfe für ein Programm erhalten?

Antwort

Mit dem Programm *man* kann man die Hilfeseiten (man pages) für Programme anzeigen lassen.

Frage 1.1.c

Wie kann man einfach nach Dateien suchen?

Antwort

Für das Suchen von Dateien, die bestimmte Kriterien erfüllen sollen, kann man das Programm *find* benutzen.

Beispiel

Um eine Datei *syscalls.c* zu suchen wechselt man in das Hauptverzeichnis des Projektes und führt den folgenden Befehl aus:

```
find . -name „syscalls.c“
```

Frage 1.1.d

In einem Projekt sind mehrere C Quelltextdateien in verschiedenen Unterverzeichnissen verteilt. Wie kann man einfach nach dem Makro *FOOBAR* in diesen Dateien suchen?

Antwort

```
find . -name „*.c“ | xargs grep „FOOBAR“
```

Mit Hilfe einer Pipe können Ein- und Ausgabe von mehreren Programmen miteinander verkettet werden.

Frage 1.1.e

Welche Schritte sind für die Erzeugung eines ausführbaren Programms aus mehreren C Quelltextdateien notwendig?

Antwort

- 1 Die Quelltextdateien werden einzeln übersetzt, daraus entstehen mehrere Objektdateien. Eine Objektdatei enthält Maschinencode, aber auch noch nicht aufgelöste Verweise auf Symbole (Funktionsaufrufe, Variablen) in anderen Objektdateien.
- 2 Alle Objektdateien werden zu einem ausführbaren Programm gelinkt. Während dem Linken werden Symbole zu realen Speicher-Adressen oder -Bereichen aufgelöst.

Frage 1.1.f

Was macht das Programm *make*?

Antwort

make vereinfacht das Erstellen von Programmen aus mehreren Quelltextdateien. Informationen über die verschiedenen Erzeugungsschritte und deren Abhängigkeiten werden in der Datei *Makefile* gespeichert.

Beispiel: Makefile

```
TARGET: dependencies
command 1
command 2
command n
```

Grundlagen der Bitarithmetik

- Bit-Shift (<< und >>)
- Bitweise Negation (\sim)
- Bitweises AND ($\&$)
- Bitweises OR ($\&$)

(Logischer) Linksshift

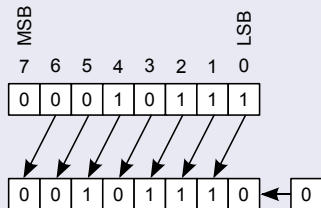


Abbildung: $00010111 \ll 1$

(Logischer) Rechtsshift

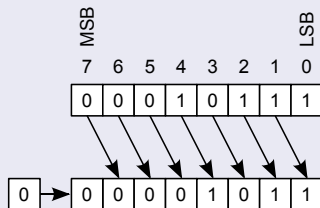


Abbildung: $00010111 \gg 1$

Bitweise Negation

Bitweise Invertierung eines Eingabewertes

AND/OR Funktion

A	B	$A \wedge B$	$A \vee B$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Bitweises AND/OR

Anwendung der AND bzw. OR Funktion bitweise auf zwei gegebene Eingabewerte

Beispiel: Bitweises AND

	10100101
&	01100011
=	00100001

Beispiel: Bitweises OR

	10100101
	01100011
=	11100111

Frage 1.2.a

Weshalb ist es manchmal notwendig einzelne Bits in einem Integer Wert zu setzen oder zu löschen?

Antwort

Hardwaregeräte definieren oft Gruppen von Speicherbereichen, in denen jedes Bit eine besondere Bedeutung hat. Durch das Setzen und Löschen von Bits kann das Hardwaregerät vom Betriebssystem gesteuert werden.

Auch im Betriebssystem selber kommen Bitgruppen vor:
In der Speicherverwaltung wird z.B. der Schreibschutz einer Seite im Hauptspeicher von einem einzelnen Bit aktiviert oder deaktiviert. Weiterhin kann Speicherplatz für die Verwaltung von Informationen gespart werden, bei einer einfachen Block-basierten Speicherverwaltung reicht ein Bit aus, um einen bestimmten Block als frei oder belegt zu kennzeichnen.

Die verschiedenen Speicherverwaltungsformen werden in einem späteren Tutorium genauer behandelt.

Frage 1.2.b

Wie kann man ein Bit eines Integer Wertes setzen?

Antwort

```
result = value | (1 << i);
```

Frage 1.2.c

Wie kann man ein Bit eines Integer Wertes löschen?

Antwort

```
result = value & ~(1 << i);
```

Frage 1.2.d

Wie kann man die Bits 2 bis 5 eines Integer Wertes erhalten?

Antwort

```
result = value & (0xF << 2);
```

Frage 1.3.a

Was ist der Unterschied zwischen einer Strategie (policy) und einem Mechanismus (mechanism)?

Antwort

Eine Strategie beschreibt, wie ein bestimmtes Ziel erreicht werden soll. Mechanismen werden eingesetzt um eine Richtlinie umzusetzen.

Beispiel: Supermarkt

Ziele

- Personalkosten minimieren
- Kunden dürfen nicht zu lange an der Kasse warten

Strategie

Solange an der Warteschlange nur wenige Kunden anstehen wird eine Kasse geöffnet. Wenn die Warteschlange zu lang werden sollte wird eine weitere Kasse geöffnet.

Mechanismus

Mit einem Mikrofon können weitere Mitarbeiter an die Kassen gerufen werden.

Frage 1.3.b

Was sind die Hauptaufgaben eines Betriebssystems?

Antwort

Abstraktion

Ein Betriebssystem soll Hardwareeigenheiten vor Anwendungen/Anwendungsprogrammierer/Benutzer verbergen.

Ressourcen-Management

Ressourcen müssen in einer „fairen“ Art und Weise zwischen Anwendungen/Benutzern verteilt werden.

Sicherheit & Schutz

Unterschiedliche Anwendungen dürfen sich nicht gegenseitig stören.
Private Daten eines Benutzers dürfen nicht ohne Genehmigung anderen Benutzern zugänglich gemacht werden.

Bereitstellung einer Umgebung für die Ausführung von Programmen

Dieser Punkt ist das zentrale Ziel von Betriebssystemen, die vorher genannten Punkte werden für die Erfüllung vorausgesetzt.

Frage 1.4.a

Was sind die Unterschiede zwischen den Prozessormodi „Kernel-Modus“ (kernel mode) und „Benutzer-Modus“ (user mode)?
Weshalb werden beide Modi gebraucht?

Antwort

Im Benutzer-Modus können nur nichtprivilegierte Befehle ausgeführt werden. Das Ausführen eines privilegierten Prozessor-Befehls führt im Benutzer-Modus zu einer Ausnahme (exception) des Typs „privileged instruction violation“. In den meisten Fällen wird bei der Ausnahmebehandlung das Programm, das die Ausnahme verursacht hat abgebrochen.

Im Kernel-Modus erlauben die meisten Prozessoren die Ausführung aller Befehle.

Manche Aktivitäten, die das System kontrollieren und verwalten, können nur im Kernel-Modus ausgeführt werden. Der Kernel-Modus ist notwendig, damit verhindert werden kann, dass Programme das System für ihre eigenen Zwecke umkonfigurieren können und damit andere Programme benachteiligen können.

Ergänzungsfrage

Was sind typische privilegierte Befehle?
Weshalb sind diese privilegiert?

Antwort

Privilegierte Befehle fallen häufig in folgende Klassen:

- Kontroll-Register verändern
- Interrupts an- oder ausschalten
- Geräte, die privilegiert sind, ansprechen

Wenn diese Befehle nicht privilegiert wären könnte ein Anwendungsprogramm kritische Systemzustände verändern, seine Privilegien erhöhen, Sicherungen umgehen oder Systemeigenschaften wie Stabilität kompromitieren.

Ergänzungsfrage

Bei einem Systemaufruf werden Parameter an den Betriebssystem-Kern übergeben. Wie kann dies umgesetzt werden?

Antwort

Die Übergabe der Parameter eines Systemaufrufs funktioniert ähnlich wie bei dem Aufruf einer Funktion in einem Anwendungsprogramm.

Parameter können deshalb in Registern, vordefinierten Speicherbereichen oder dem Stack übergeben werden.

Die Übergabe via Register ist der schnellste Weg, allerdings ist die Anzahl der Register limitiert.

Bei der Übergabe via Hauptspeicher können mehr Parameter übergeben werden, allerdings kommen zusätzliche Speicherzugriffe dazu.

Frage 1.4.b

Was sind die Grundlagen und Vorteile einer Speicherhierarchie?

Antwort

Eine Speicherhierarchie bezeichnet die Anordnung von Arbeits- und Datenspeichern verschiedener Technologie nach sinkender Zugriffsgeschwindigkeit und steigender Speicherkapazität in einer Rechnerarchitektur.

Der Hauptvorteil ist die bessere Organisation des Speichers, es können z.B. zur Optimierung der Zugriffszeiten auf häufig benötigte Daten Speicherinhalte innerhalb der Hierarchie verschoben werden.

Beispiel: Speicherhierarchie

- 1 L1-Cache
- 2 L2-Cache
- 3 Arbeitsspeicher
- 4 Massenspeicher

Frage 1.4.b

Welches typische Programmverhalten deckt sich mit den Vorteilen einer Speicherhierarchie?

Antwort

Programme besitzen häufig die folgenden Lokalitätseigenschaften:

- Zeitliche Lokalität: In einem gewissen Zeitabschnitt werden Bereiche der Befehls- oder Datenmenge häufig durchlaufen.
- Örtliche Lokalität: Befehls- oder Datenzugriffe finden auf einem kleinen Bereich des gesamten Adressraums statt.

Beispiel: Stack für lokale Variablen

Beim Aufruf einer Funktion wird der Stack für die lokalen Variablen der Funktion erweitert. Diese lokale Variablen

- liegen im Speicher nebeneinander (\Rightarrow Örtliche Lokalität)
- werden häufig verändert (\Rightarrow Zeitliche Lokalität)

Frage 1.4.c

Die kleinste Verwaltungseinheit innerhalb des Caches eines Prozessors ist ein Cache-Block (oder Cache-Line). Dieser Cache-Block ist häufig mehrere Wörter groß.

Weshalb ist der Cache in diese Cache-Blöcke geteilt?

Antwort

Aufgrund der Lokalitätseigenschaften von Programmen ist es geschickt, große Mengen an Daten gleichzeitig neu im Cache zu speichern.

Zusätzlich können bei großen Blockgrößen Daten schneller aus dem Hauptspeicher übertragen werden als mit vielen Zugriffen mit kleinen Blockgrößen.

Einführung Cache-Organisation

(weitere Informationen in TI II bzw. Rechnerorganisation)

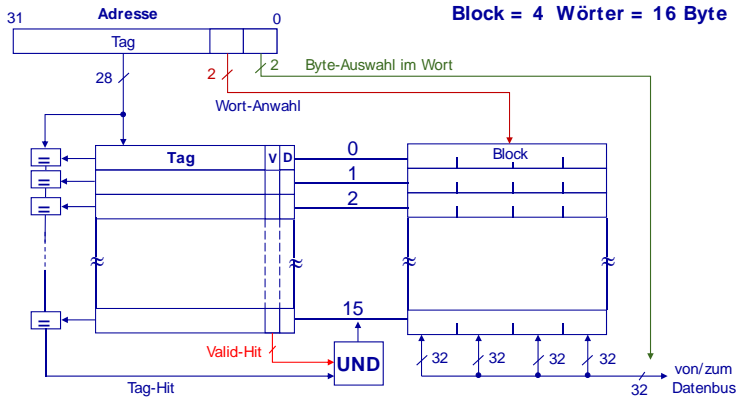
- Cache-Blöcke werden in Cache-Zeilen zusammengefasst
- Cache-Zeilen werden in Sätzen geordnet

Für die Anordnung gibt es verschiedene Ansätze.

Beispiel: Vollassoziativer Cache

Kapazität: 256 Byte

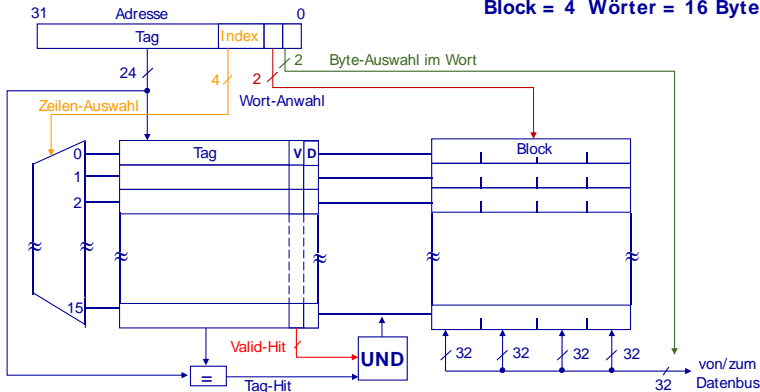
Block = 4 Wörter = 16 Byte



Beispiel: Direct-mapped-Cache

Kapazität: 256 Byte

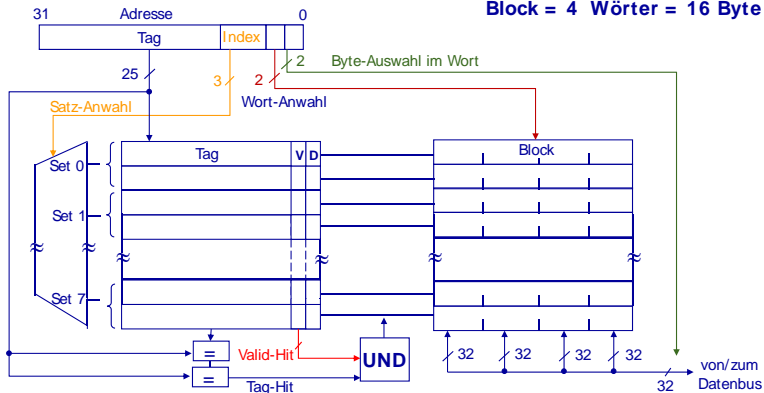
Block = 4 Wörter = 16 Byte



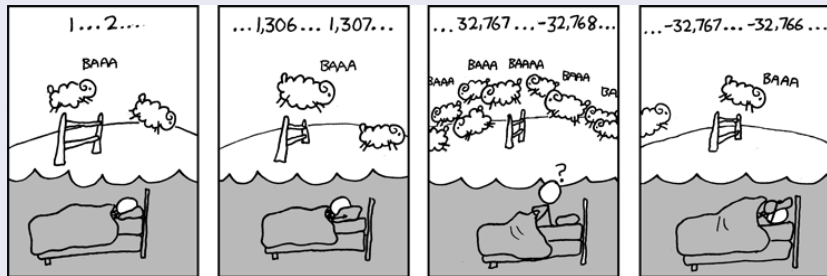
Beispiel: 2-way-set-assoziativer Cache

Kapazität: 256 Byte

Block = 4 Wörter = 16 Byte



Fragen & Kommentare?



xkcd: Can't Sleep