Betriebssysteme (BS)

Probeklausur

Olaf Spinczyk und **Alexander Lochmann**
Arbeitsgruppe Eingebettete Systemsoftware

Lehrstuhl für Informatik 12
TU Dortmund

http://ess.cs.uni-dortmund.de/

Ankündigung

- Rechnerübung findet nochmal zur LCT-Vorbereitung statt:
  - Mittwoch, 08. Juli
  - Montag, 13. Juli

Ablauf

- Probeklausur (45 Minuten)
- Besprechung der Aufgaben
- Auswertung
- Weitere Hinweise zur Vorbereitung

Probeklausur

... in (fast) allen Belangen realistisch:

- Art der Aufgaben
  - Auswahl aus dem **gesamten** Inhalt der Veranstaltung
    - Betriebssystemgrundlagen **und** UNIX-Systemprogrammierung in C
    - alle Vorlesungen und Übungen sind relevant
- Umfang
  - kürzer als das „Original“: ca. 40–45 (statt 60) Minuten
- Durchführung
  - **keine Hilfsmittel** erlaubt (keine Spickzettel, Bücher, ...)
  - bitte **still arbeiten**
  - jeder für sich
- Die Klausur wird **nicht** eingesammelt.

1 – Prozesse und Scheduling

- a) Prozesserzeugung
  - Was wird ausgegeben?

```
int pferd;

void* erzeugePferd(void* param) {
        pferd++;
        printf("%d\n",pferd);
        return NULL;
}

int main(void) {
        pferd = 42;
        pid_t ret;

        if (fork() == 0) {
                erzeugePferd(NULL);
        } else {
                erzeugePferd(NULL);
        }
        return 0;
}
```

43 43

```
int pferd;

void* erzeugePferd(void* param) {
        pferd++;
        printf("%d\n",pferd);
        pthread_exit(NULL);

        return NULL;
}

int main(void) {
        pferd = 42;
        pthread_t id;

        pthread_create(...);
        erzeugePferd(NULL);
        pthread_exit(NULL);
        return 0;
}
```

43 44

1 – Prozesse und Scheduling

- a) Prozesserzeugung
  - Wieso ist die Ausgabe unterschiedlich?

- fork()
  - Erzeugt neuen Prozess
  - Kopiert Adressraum

    → eigene Version von Data, BSS, Heap und Stack

Jeder hat sein eigenes „pferd“

- pthread_create()
  - Erzeugt neuen Thread
  - Legt eigenen Stack an
  - Teilt sich Data, BSS und Heap

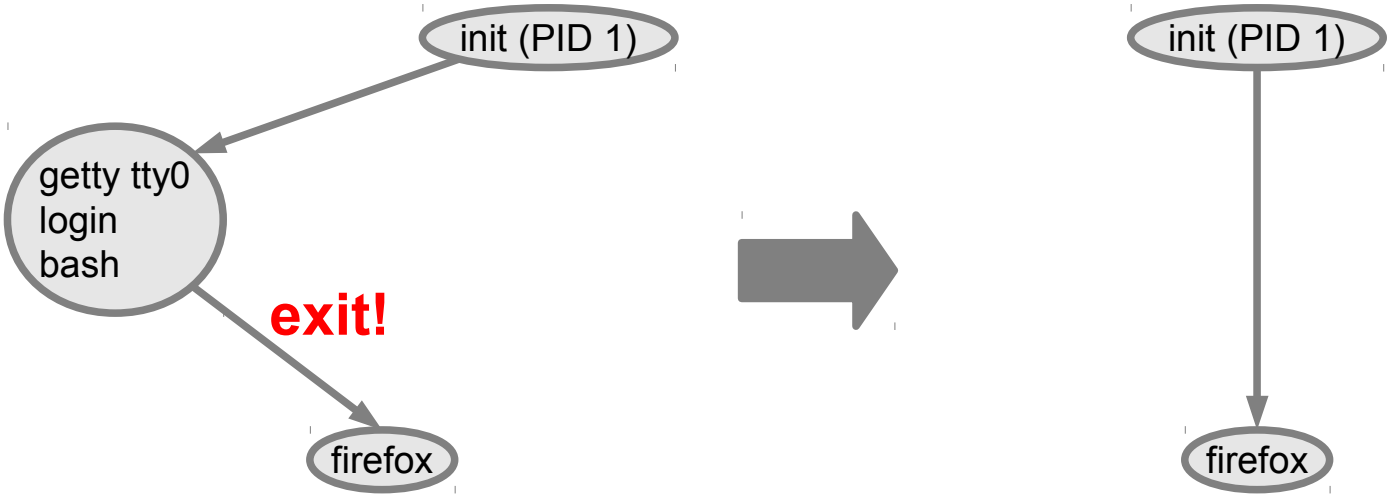
Beide teilen sich ein „pferd“

1 – Prozesse und Scheduling

- b) Prozesse
  - 1) Was geschieht mit den Kindprozessen, wenn der Elternprozess terminiert?

(engl. *„orphan processes“*)

- Ein UNIX-Prozess wird zum Waisenkind, wenn sein Elternprozess terminiert.
- Was passiert mit der Prozesshierarchie?



init (PID 1) adoptiert alle verwaisten Prozesse.
So bleibt die Prozesshierarchie intakt.

1 – Prozesse und Scheduling

- b) Prozesse
  - 2) Was ist der *copy-on-write* Mechanismus? Wo kommt er im Kontext von Prozessen zum Einsatz?

- *Copy-on-write*
  - Zwei (oder mehr) Entitäten arbeiten auf demselbem Datum
  - Erst bei einem schreibendem Zugriff werden Kopien erstellt

- Einsatz im Kontext von Prozessen
  - Verwendung bei fork()
  - Erlaubt schnelles Erzeugen von Prozessen
  - Beide (Vater und Kind) arbeiten zunächst auf denselbem BSS-, Daten-, und Heap-Segment – bis einer der beiden etwas verändert

1 – Prozesse und Scheduling

- c) Scheduling

J	N	
	X	Bei Round Robin bekommt der nächste Prozess die restliche Zeitscheibe des Vorgängers.
X		Shortest Remaining Time First (SRTF) kann Prozesse zum Verhungern bringen.
X		Bei Round Robin ist die CPU-Zeit zu gunsten CPU-lastiger Prozesse ungleich verteilt.
	X	Bei *präemptivem Scheduling* wird davon ausgegangen, dass Prozesse freiwillig die CPU abgeben.
	X	First-Come First-Served (FCFS) ist optimal geeignet bei einem Mix von CPU- und E/A-lastigenProzessen.

- SRTF → Übungsaufgabe A1, Theorieaufgabe 1
- E/A-lastige Prozesse geben die CPU sofort ab und kommen ans Ende der *ready*-Liste
- Präemptives Scheduling soll eine Monopolisierung der CPU verhindern
- E/A-lastige Prozesse werden benachteiligt

2 – Synchronisation und Verklemmung

- Die Funktionen producer() und consumer() in folgendem Pseudocode-Abschnitt werden potentiell gleichzeitig ausgeführt, wobei producer() Elemente erzeugt und consumer() diese verbraucht. Synchronisieren Sie die beiden Funktionen mittels einseitiger Synchronisation und beachten Sie dabei, dass die Warteschlange zwar beliebig viele Elemente aufnehmen kann, die Operationen enqueue() und dequeue() aber selbst kritisch sind und nicht gleichzeitig ausgeführt werden dürfen. Der consumer() soll nur Elemente aus der Warteschlange entfernen, wenn diese nicht leer ist!

  Legen Sie dazu geeignet benannte Semaphore an, initialisieren Sie diese, und setzen Sie an den freien Stellen Semaphor-Operationen (P, V bzw. wait, signal) ein (z.B. P(MeinSemaphor)). (P(), V(), produce_element(), consume_element(), enqueue() und dequeue() können als gegeben angesehen werden und müssen nicht implementiert werden!)

2 – Synchronisation und Verklemmung

- a) Namen und Initialwerte der Semaphore:
  - `mutex` – Wert=1
  - `elements` - Wert=0

```
producer () {
  while (1) {
   Element e = produce_element ();
   P(mutex);
   enqueue ( e );
   V(mutex);
   V(elements);
  }
}
```

```
consumer() {
    while (1) {
       P(elements);
       P(mutex);
       Element e = dequeue();
       V(mutex);
       consume_element(e);
    }
}
```

2 – Synchronisation und Verklemmung

- b) Erläutern Sie, was man unter sogenannten *Race Conditions* versteht und wie man diese verhindern kann.
  - *Race Conditions* entstehen, wenn mehrere Prozesse auf die selben Daten zugreifen und mindestens ein Prozess diese manipuliert.

2 – Synchronisation und Verklemmung

- c) Nennen und erläutern Sie zwei Bedingungen, die notwendig sind, damit eine Verklemmung auftreten kann.

1) Exklusive Belegung von Betriebsmitteln (***„mutual exclusion“***)
   - die umstrittenen Betriebsmittel sind nur unteilbar nutzbar

2) Nachforderung von Betriebsmitteln (***„hold and wait“***)
   - die umstrittenen Betriebsmittel sind nur schrittweise belegbar

3) Kein Entzug von Betriebsmitteln (***„no preemption“***)
   - Die umstrittenen Betriebsmittel sind nicht rückforderbar

4) Zirkuläres Warten (***„circular wait“***)
   - Eine geschlossene Kette wechselseitig wartender Prozesse

2 – Synchronisation und Verklemmung

- d) Nennen Sie eine Möglichkeit zur Verklemmungsvorbeugung (*deadlock prevention*)
  - **indirekte Methoden** entkräften eine der Bedingungen 1–3
    - nicht-blockierende Verfahren verwenden
    - Betriebsmittelanforderungen unteilbar (atomar) auslegen
    - Betriebsmittelentzug durch **Virtualisierung** ermöglichen
      - virtueller Speicher, virtuelle Geräte, virtuelle Prozessoren
  - **direkte Methoden** entkräften Bedingung 4
    - lineare/totale Ordnung von Betriebsmittelklassen einführen:
      - Betriebsmittel B_i ist nur dann erfolgreich vor B_j belegbar, wenn *i* linear vor *j* angeordnet ist (d.h. $i < j$).

3 – Speicherverwaltung + virt. Speicher

- a) Adressabbildung

Seitennummer	Startadresse
0	$F000_{16}$
1	3000_{16}
2	8000_{16}
3	1000_{16}
4	$C000_{16}$
5	2000_{16}
6	4000_{16}
7	$B000_{16}$
...	...
15	5000_{16}

logische Adresse: $6AB1_{16}$

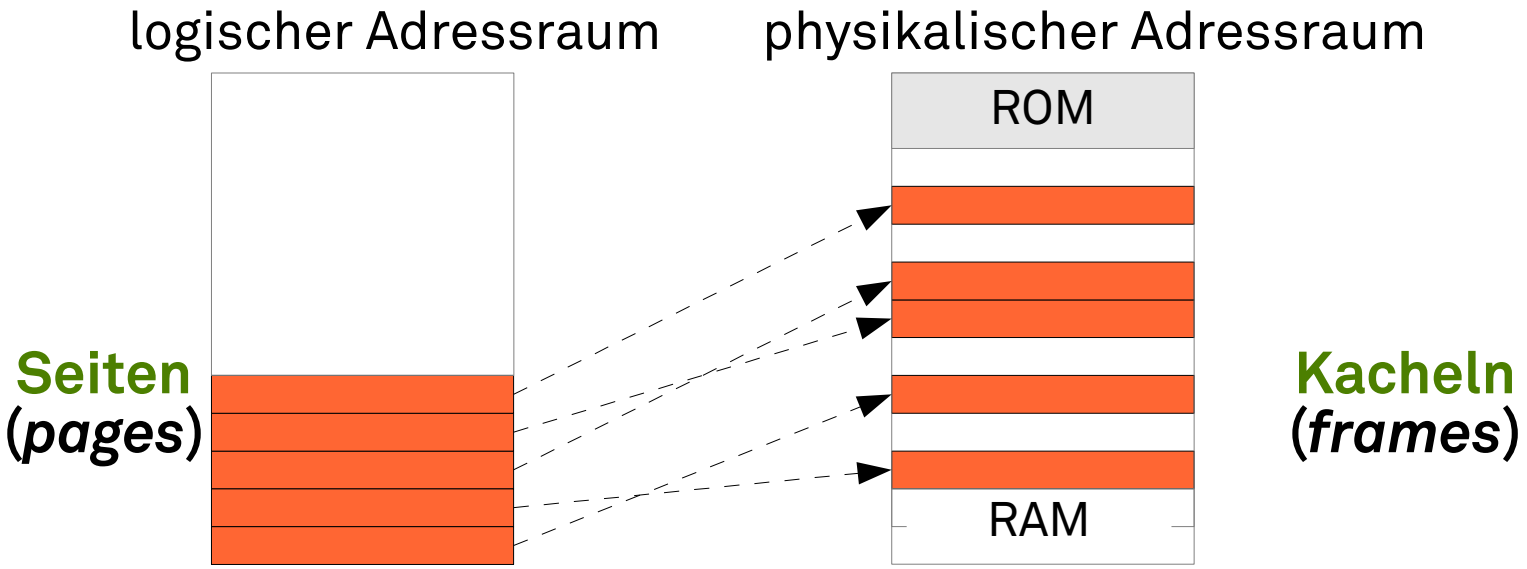
$\rightarrow$ physikalische Adresse:

logische Adresse: $F1B7_{16}$

$\rightarrow$ physikalische Adresse:

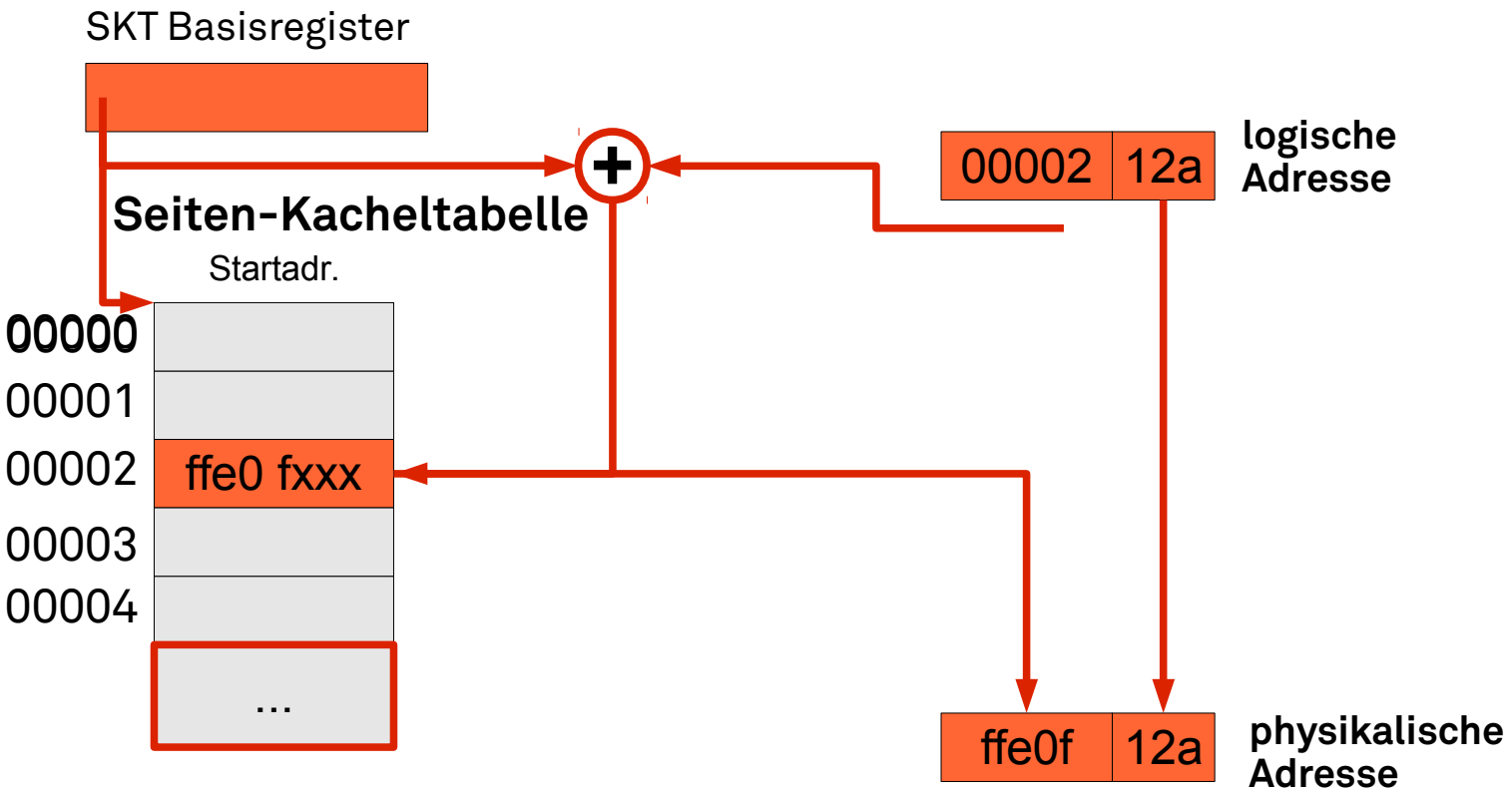
Seitenadressierung (*paging*)

- Einteilung des logischen Adressraums in gleichgroße Seiten, die an beliebigen Stellen im physikalischen Adressraum liegen können
  - Lösung des Fragmentierungsproblems
  - keine Kompaktifizierung mehr nötig
  - Vereinfacht Speicherbelegung und Ein-/Auslagerungen



MMU mit Seiten-Kacheltabelle

- Tabelle setzt Seiten in Kacheln um



3 – Speicherverwaltung + virt. Speicher

- a) Adressabbildung

Seitennummer	Startadresse
0	$F000_{16}$
1	3000_{16}
2	8000_{16}
3	1000_{16}
4	$C000_{16}$
5	2000_{16}
6	4000_{16}
7	$B000_{16}$
...	...
15	5000_{16}

4000 OR AB1

logische Adresse: $6AB1_{16}$

→ physikalische Adresse: 4AB1

5000 OR 1B7

logische Adresse: $F1B7_{16}$

→ physikalische Adresse: 51B7

3 – Speicherverwaltung + virt. Speicher

- b) LRU (Seitenersetzung)

Referenzfolge		3	1	2	4	2	1	5	3	2
	Kachel 1	3	3	3	4	4	4	5	5	5
	Kachel 2		1	1	1	1	1	1	1	2
	Kachel 3			2	2	2	2	2	3	3
Kontrollzustände	Kachel 1	0	1	2	0	1	2	0	1	2
	Kachel 2		0	1	2	3	0	1	2	0
	Kachel 3			0	1	0	1	2	0	1

4 - Ein-/Ausgabe und Dateisysteme

- Gegeben sei ein Plattenspeicher mit 16 Spuren. Der jeweilige I/O-Scheduler bekommt immer wieder Leseaufträge für eine bestimmte Spur. Die Leseaufträge in L_1 sind dem I/O-Scheduler bereits bekannt. Nach **einem** bearbeiteten Auftrag erhält er die Aufträge in L_2. Nach **weiteren drei** (d.h. nach insgesamt vier) bearbeiteten Aufträgen erhält er die Aufträge in L_3. Zu Beginn befindet sich der Schreib-/Lesekopf über Spur 0.

$$L_1=\{4,7,11,3\} \quad L_2=\{2,13,1\} \quad L_3=\{15,5,6\}$$

4 - Ein-/Ausgabe und Dateisysteme

$$L_1 = \{4,7,11,3\} \quad L_2 = \{2,13,1\} \quad L_3 = \{15,5,6\}$$

Sofort bekannt

Nach 1 Ops bekannt

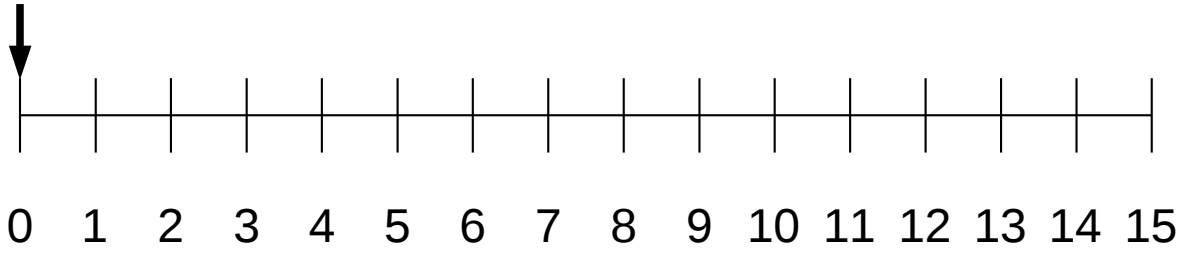
Nach 4 Ops bekannt

- Bitte tragen Sie hier die Reihenfolge der gelesenen Spuren für einen I/O-Scheduler, der nach der **First In First Out (FIFO)** Strategie arbeitet, ein:

4 - Ein-/Ausgabe und Dateisysteme

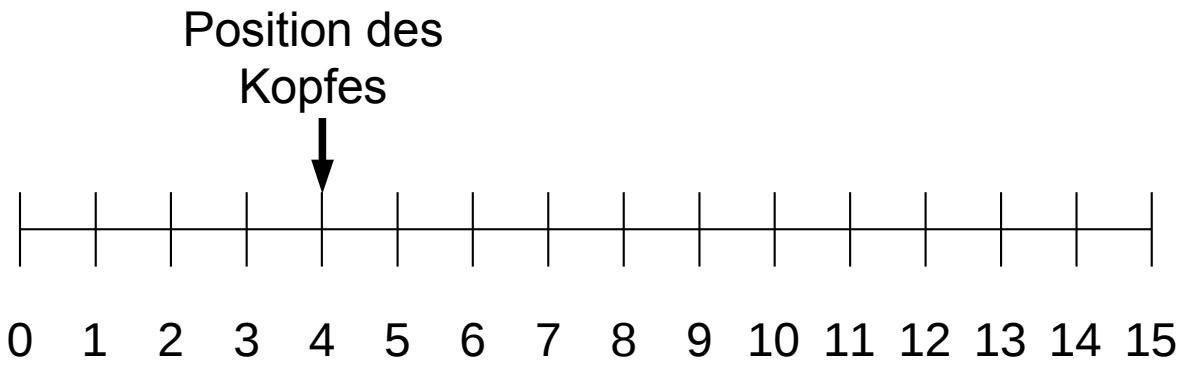
T = 0 I/O-Anfragen:
4, 7, 11, 3

Position des Kopfes



4 - Ein-/Ausgabe und Dateisysteme

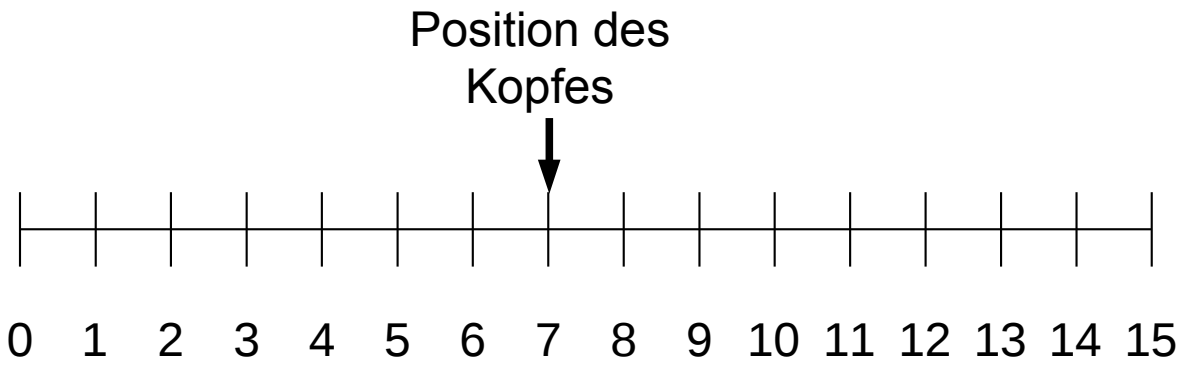
T = 1 I/O-Anfragen:
7, 11, 3, 2, 13, 1



4									

4 - Ein-/Ausgabe und Dateisysteme

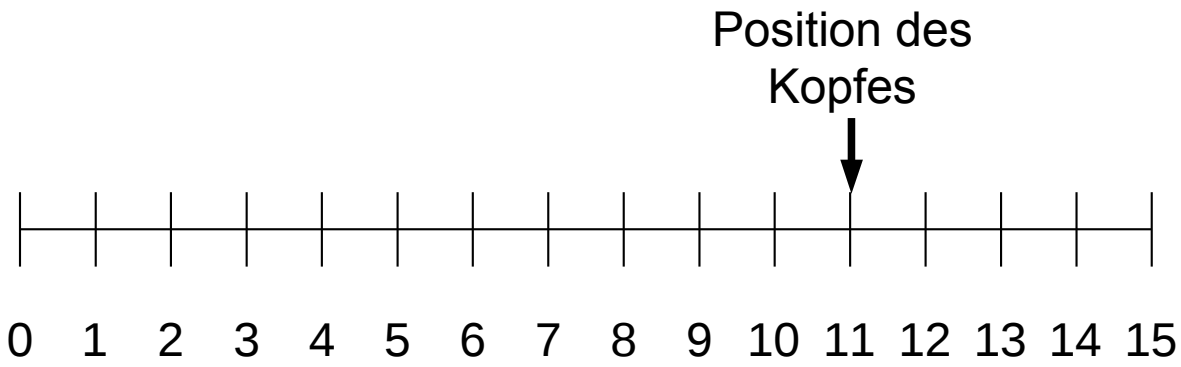
T = 2 I/O-Anfragen:
11, 3, 2, 13, 1



4	7								

4 - Ein-/Ausgabe und Dateisysteme

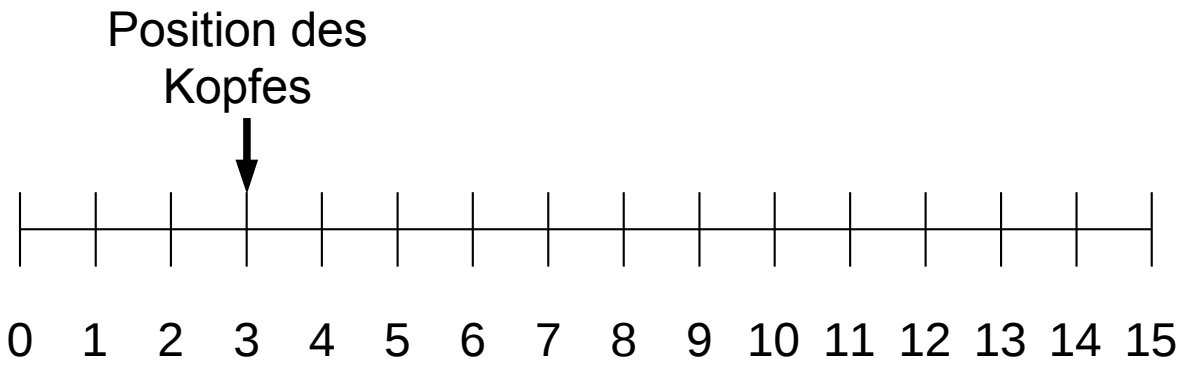
T = 3 I/O-Anfragen:
3, 2, 13, 1



4	7	11							

4 - Ein-/Ausgabe und Dateisysteme

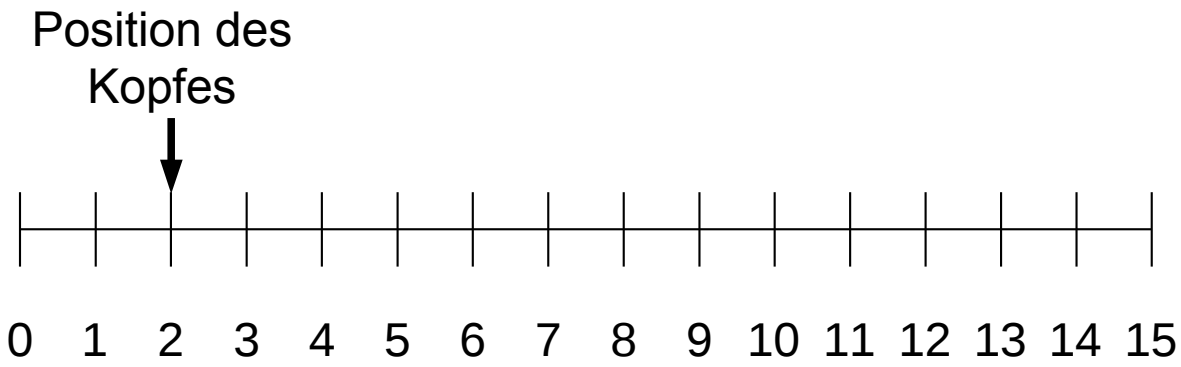
T = 4 I/O-Anfragen:
2, 13, 1, 15, 5, 6



4	7	11	3						

4 - Ein-/Ausgabe und Dateisysteme

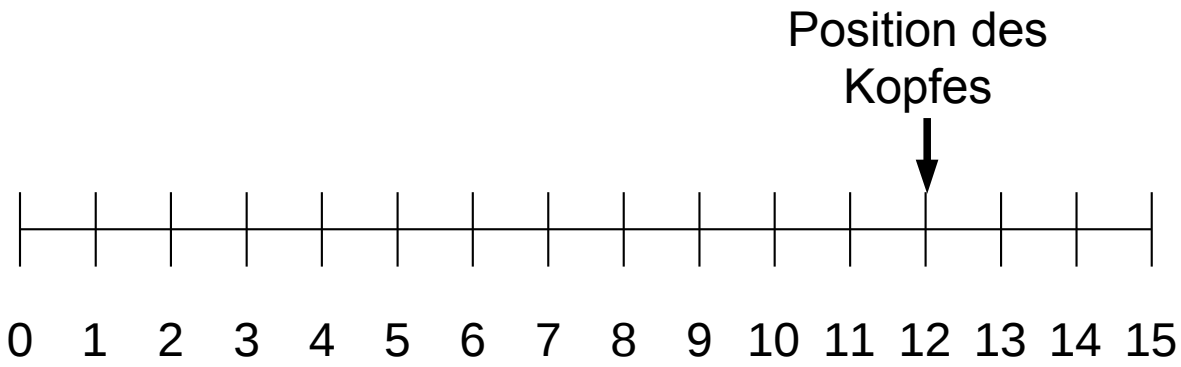
T = 5 I/O-Anfragen:
13, 1, 15, 5, 6



4	7	11	3	2					

4 - Ein-/Ausgabe und Dateisysteme

T = 6 I/O-Anfragen:
1, 15, 5, 6

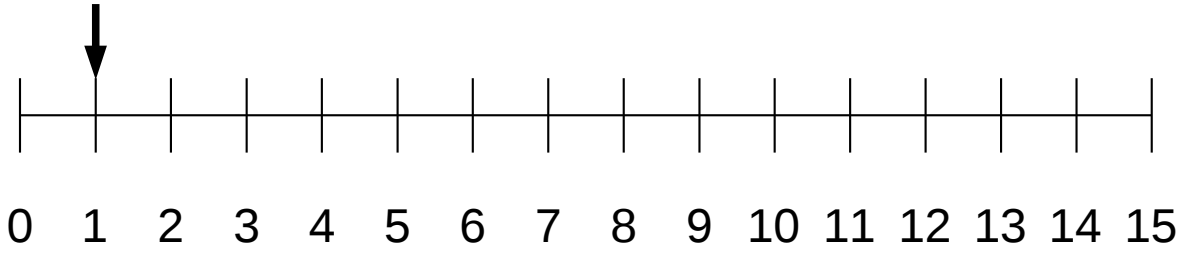


4	7	11	3	2	13				

4 - Ein-/Ausgabe und Dateisysteme

T = 7 I/O-Anfragen:
15, 5, 6

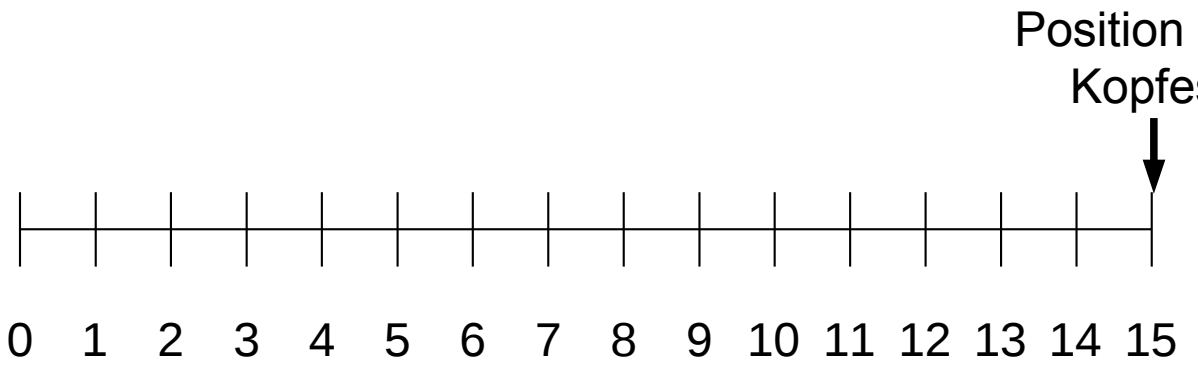
Position des Kopfes



4	7	11	3	2	13	1			

4 - Ein-/Ausgabe und Dateisysteme

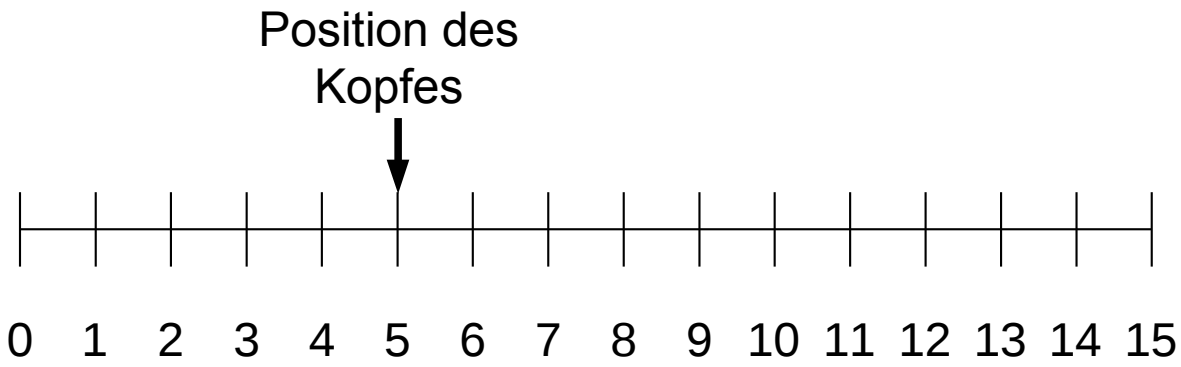
T = 8 I/O-Anfragen:
5, 6



4	7	11	3	2	13	1	15		

4 - Ein-/Ausgabe und Dateisysteme

T = 9 I/O-Anfragen:
6

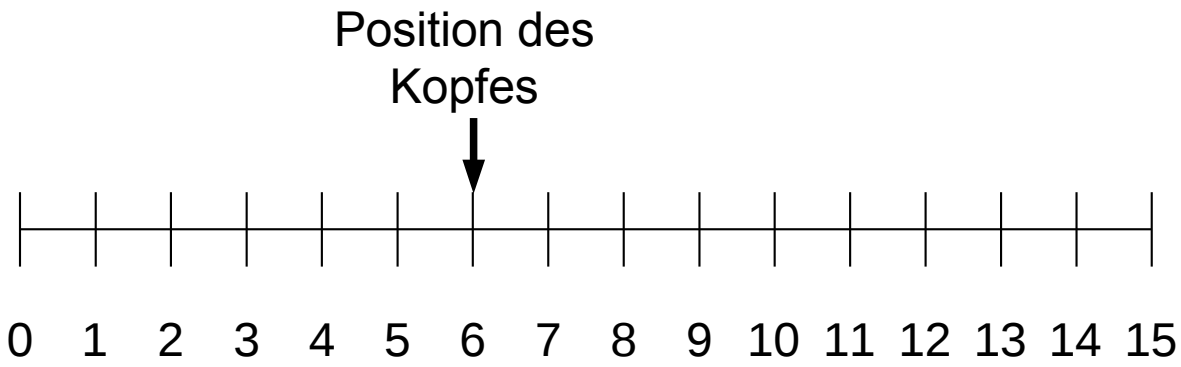


4	7	11	3	2	13	1	15	5	

4 - Ein-/Ausgabe und Dateisysteme

T = 10

I/O-Anfragen:
6



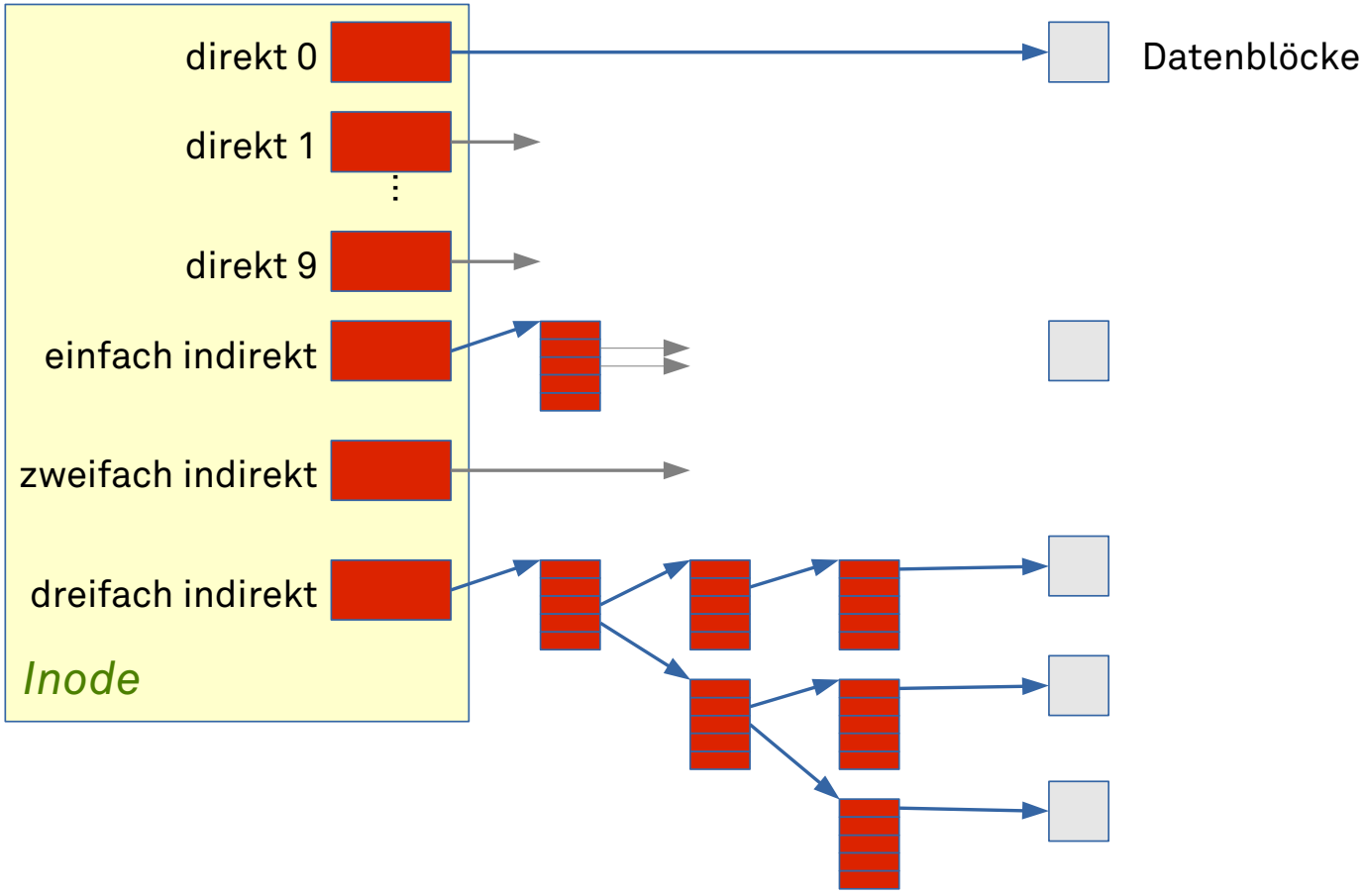
4	7	11	3	2	13	1	15	5	6

4 - Ein-/Ausgabe und Dateisysteme

- b) Nennen Sie je einen Vorteil und einen Nachteil der kontinuierlichen Datenspeicherung
  - Vorteile:
    - Zugriff auf alle Blöcke mit minimaler Positionierzeit des Schwenkarms
    - Schneller direkter Zugriff auf bestimmter Dateiposition
    - Einfacher Einsatz bei nicht-beschreibbaren Medien (CD, etc)
  - Nachteile:
    - **Finden des freien Platzes** auf der Festplatte (Menge aufeinanderfolgender und freier Plattenblöcke)
    - **Fragmentierungsproblem** (Verschnitt: nicht nutzbare Plattenblöcke; siehe auch Speicherverwaltung)
    - **Größe bei neuen Dateien** oft nicht im Voraus bekannt
    - **Erweitern** ist problematisch
    - **Umkopieren**, falls kein freier angrenzender Block mehr verfügbar

4 - Ein-/Ausgabe und Dateisysteme

- c) Indizierte Speicherung



4 - Ein-/Ausgabe und Dateisysteme

- c) Indizierte Speicherung
  - 1. Nennen Sie beispielhaft ein Dateisystem, bei dem Dateien in der dargestellten Weise abgelegt werden.UNIX-, Linux-Dateisysteme, EXT2,3,4 Filesystem

- UNIX-, Linux-Dateisysteme, wie z.B. EXT2,3,4 Filesystem

4 - Ein-/Ausgabe und Dateisysteme

- c) Indizierte Speicherung
  - 2. Ein hypothetisches Dateisystem verwendet Inodes wie oben dargestellt, nur **ohne** Dreifach-Indirektion.
    Wie lässt sich die maximale Dateigröße für dieses Dateisystem berechnen, wenn die Blockgröße 1024 Bytes beträgt und für die Speicherung eines Blockverweises 4 Bytes benötigt werden.

    Hinweis: Es ist nicht erforderlich die Zahl auszurechnen.
    Beschreiben Sie den Rechenweg Schritt für Schritt oder geben Sie eine Formel an.

4 - Ein-/Ausgabe und Dateisysteme

- B: Maximale Anzahl der Blöcke
- BG: Blockgröße
- D: Maximale Dateigröße (gesucht)
- I: Anzahl der Verweise in indirekten Blöcken

Jeder Blockverweis ist 4 Byte groß

$$I = BG/4 = 1024/4$$

Direkte Verweise

Einfache Indirektion

Zweifache Indirektion

$$D = BG * B = BG * (10 + I + I * I)$$
$$= 1024\,Bytes * (10 + 256 + 256 * 256)$$
$$= 67381248\,Bytes \quad = 65802\,KiBytes$$

4 - Ein-/Ausgabe und Dateisysteme

- c) Indizierte Speicherung
  - 3) Nennen Sie einen Vorteil und einen Nachteil der indizierten Speicherung im Vergleich zur kontinuierlichen Speicherung.

- Vorteil:
  - Auch große Dateien lassen sich so addressieren
- Nachteil:
  - Bei großen Dateien müssen mehrere Blöcke gelesen werden

Auswertung

- Bitte schnell einmal die Punkte zusammenzählen …
- Notenspiegel:

Punkte	Note
38,5–45	1
33,5–38	2
28–33	3
22,5–27,5	4
0–22	5

Weitere Hinweise zur Vorbereitung

- Inhalt der Folien lernen
  - Klassifizieren: Was muss ich **lernen**? Was muss ich **begreifen**?
- Übungsaufgaben verstehen, C und UNIX „können“
  - ASSESS-System bleibt mindestens bis zur Klausur offen
    - bei Fragen zur Korrektur melden
  - Am besten die Aufgaben noch einmal lösen
  - Optionale Zusatzaufgaben bearbeiten
- Literatur zur Lehrveranstaltung durchlesen
- BS-Forum nutzen

Empfohlene Literatur

[1] A. Silberschatz et al. *Operating System Concepts*. Wiley, 2004. ISBN 978-0471694663

[2] A. Tanenbaum: Modern Operating Systems (2nd ed.). Prentice Hall, 2001. ISBN 0-13-031358-0

[3] B. W. Kernighan, D. M. Ritchie. *The C Programming Language*. Prentice-Hall, 1988. ISBN 0-13-110362-8 (paperback) 0-13-110370-9 (hardback)

[4] R. Stevens, *Advanced Programming in the UNIX Environment*, Addison-Wesley, 2005. ISBN 978-0201433074

Viel Erfolg bei der Klausur!