GBS Übung

Woche 12

Aufgabe 1 Vorbereitung

Vor dieser Übung sollten Sie...

- sich mit der Verwaltung von Daten mittels i-nodes und FATs auseinandergesetzt haben.
- ein grunsätzliches Verständnis für die Inhalte einer i-node haben.

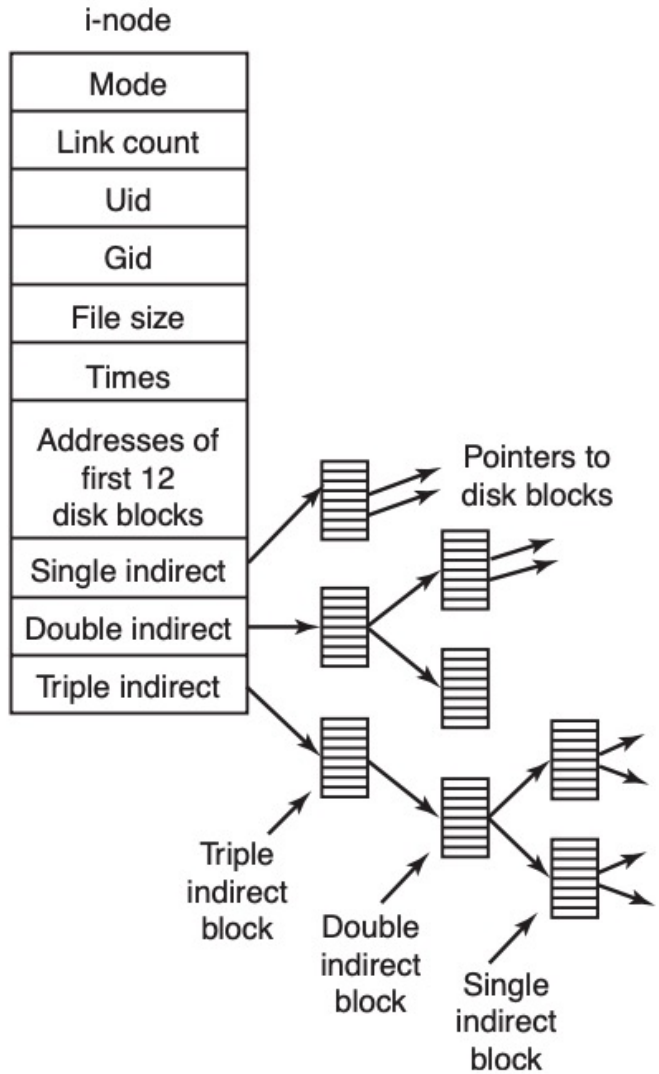


Abbildung 1: I-node (Quelle: Tanenbaum, Andrew S., and Herbert Bos. Modern operating systems. Pearson, 2015)

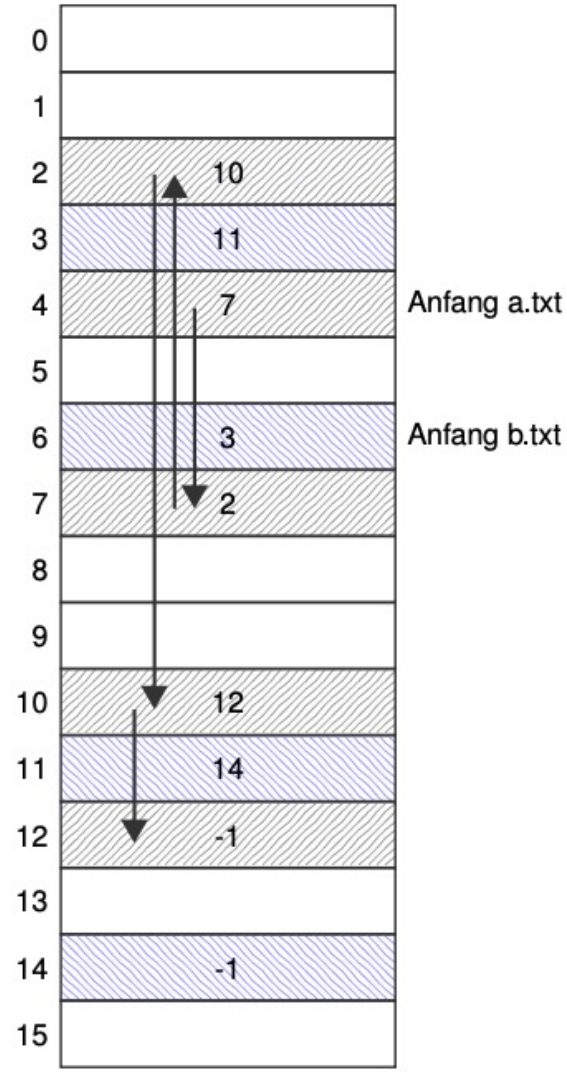


Abbildung 2: File Allocation Table (FAT)

Aufgabe 2 Maximale Dateigröße unter UNIX

Der Hintergrundspeicher ist in Blöcke aufgeteilt. Nehmen Sie im Folgenden an, ein Block habe eine Größe von 1 KiB und die Adresse eines Blocks benötige 32 Bit. Welche Größe (in Byte) kann eine Datei maximal haben, wenn im i-node der Datei neben den single-, double- und triple-indirect-Verweisen noch 12 Datenblöcke direkt referenziert werden? Begründen Sie Ihre Antwort.

Aufgabe 2 Maximale Dateigröße unter UNIX

Der Hintergrundspeicher ist in Blöcke aufgeteilt. Nehmen Sie im Folgenden an, ein Block habe eine Größe von 1 KiB und die Adresse eines Blocks benötige 32 Bit. Welche Größe (in Byte) kann eine Datei maximal haben, wenn im i-node der Datei neben den single-, double- und triple-indirect-Verweisen noch 12 Datenblöcke direkt referenziert werden? Begründen Sie Ihre Antwort.

Die Anzahl der möglichen Blöcke einer Datei ergibt sich aus der Summe der direkt, single indirect, double indirect und triple indirect adressierbaren Blöcke.

Eine Adresse hat 32 Bit, also 4 Byte. Somit lassen sich bei einer Blockgröße von 1KiB $\lfloor \frac{1024B}{4B} \rfloor = 256$ Adressen auf einem Block speichern.
Ein single-indirect Verweis kann somit 256 Blöcke adressieren, ein double-indirect kann 256 Blöcke adressieren die jeweils 256 Datenblöcke adressieren können, usw..

Es ergibt sich bei n = 12 folgende maximale Anzahl von Blöcken für eine Datei:

Direct	12 Blöcke
Single indirect	256 Blöcke
Double indirect	$256 \cdot 256 = 65536$ Blöcke
Triple indirect	$256 \cdot 256 \cdot 256 = 16777216$ Blöcke
	$= 16843020$ Blöcke

Insgesamt sind 16843020 Blöcke durch eine i-node und ihre Verweise adressierbar. Bei einer Blockgröße von 1 KiB ergibt sich somit als maximale Größe einer Datei:
$16843020 \cdot 1\ \text{KiB} \approx 16448\ \text{MiB} \approx 16{,}06\ \text{GiB}$

In der Praxis haben die Blöcke übrigens meistens 4 KiB, sodass eine Datei $(12 + 1024 + 1024^2 + 1024^3) \cdot 4096$ Byte ≈ 4 TiB haben kann.

Aufgabe 3 Vergleich: i-nodes und FAT

Auf einer 16 GiB-Festplatte werden Dateien unter UNIX mittels i-nodes gespeichert. Eine i-node verfüge im Folgenden über 12 direkte Referenzen auf Datenblöcke, einen einfach indirekten Verweis und einen zweifach indirekten Verweis. Die Blockgröße betrage 1024 Bytes und die Adresslänge 32 Bit.

a)* Wie viele Blöcke Verwaltungsaufwand benötigt eine 250 KiB-Datei?

Aufgabe 3 Vergleich: i-nodes und FAT

Auf einer 16 GiB-Festplatte werden Dateien unter UNIX mittels i-nodes gespeichert. Eine i-node verfüge im Folgenden über 12 direkte Referenzen auf Datenblöcke, einen einfach indirekten Verweis und einen zweifach indirekten Verweis. Die Blockgröße betrage 1024 Bytes und die Adresslänge 32 Bit.

a)* Wie viele Blöcke Verwaltungsaufwand benötigt eine 250 KiB-Datei?

1. 250 KiB Nutzdaten benötigen 250 Datenblöcke à 1 KiB.
2. 12 Blöcke werden direkt über die i-node adressiert. Diese liegt (neben anderen i-nodes) ebenfalls auf einem Datenblock.
3. Die restlichen 238 Blöcke werden mittels single-indirect Verweisen verwaltet. Pro Datenblock sind hierbei jeweils

$$\frac{1024\ \text{Byte}}{32\ \text{bit}} = 256$$

   Verweise möglich.
4. Insgesamt sind also 2 Blöcke Verwaltungsaufwand und 250 Blöcke für Nutzdaten nötig.

Aufgabe 3 Vergleich: i-nodes und FAT

Auf einer 16 GiB-Festplatte werden Dateien unter UNIX mittels i-nodes gespeichert. Eine i-node verfüge im Folgenden über 12 direkte Referenzen auf Datenblöcke, einen einfach indirekten Verweis und einen zweifach indirekten Verweis. Die Blockgröße betrage 1024 Bytes und die Adresslänge 32 Bit.

b)* Welchen Umfang würde eine FAT (File Allocation Table) für die obige Festplatte haben? Verwenden Sie die kleinstmögliche Anzahl von Bits für die Blocknummern.

Aufgabe 3 Vergleich: i-nodes und FAT

Auf einer 16 GiB-Festplatte werden Dateien unter UNIX mittels i-nodes gespeichert. Eine i-node verfüge im Folgenden über 12 direkte Referenzen auf Datenblöcke, einen einfach indirekten Verweis und einen zweifach indirekten Verweis. Die Blockgröße betrage 1024 Bytes und die Adresslänge 32 Bit.

b)* Welchen Umfang würde eine FAT (File Allocation Table) für die obige Festplatte haben? Verwenden Sie die kleinstmögliche Anzahl von Bits für die Blocknummern.

Die File Allocation Table ist ein Array, das für jeden Block der Festplatte einen Eintrag besitzt. Die Festplatte hat 2^{24} Blöcke der Größe 1 KiB. Daraus folgt, dass die Tabelle 2^{24} Einträge der Größe 24 bit (3 Byte) verwalten muss. $\Rightarrow$ Die FAT ist $2^{24} \cdot 24$ bit = $2^{24} \cdot 3$ Bytes = 48 MiB groß.

Aufgabe 3 Vergleich: i-nodes und FAT

Auf einer 16 GiB-Festplatte werden Dateien unter UNIX mittels i-nodes gespeichert. Eine i-node verfüge im Folgenden über 12 direkte Referenzen auf Datenblöcke, einen einfach indirekten Verweis und einen zweifach indirekten Verweis. Die Blockgröße betrage 1024 Bytes und die Adresslänge 32 Bit.

c) Welches der beiden Verfahren (i-nodes, FAT) ist bei der Verwaltung von wenigen kleinen Dateien im Vorteil? Warum? Begründen Sie Ihre Antwort (denken Sie an den Hauptspeicherbedarf).

Aufgabe 3 Vergleich: i-nodes und FAT

Auf einer 16 GiB-Festplatte werden Dateien unter UNIX mittels i-nodes gespeichert. Eine i-node verfüge im Folgenden über 12 direkte Referenzen auf Datenblöcke, einen einfach indirekten Verweis und einen zweifach indirekten Verweis. Die Blockgröße betrage 1024 Bytes und die Adresslänge 32 Bit.

c) Welches der beiden Verfahren (i-nodes, FAT) ist bei der Verwaltung von wenigen kleinen Dateien im Vorteil? Warum? Begründen Sie Ihre Antwort (denken Sie an den Hauptspeicherbedarf).

Bei wenigen kleinen Dateien ist das Unix Dateisystem (i-nodes) im Vorteil. Es müssen nur die i-nodes der aktuell geöffneten Dateien im Speicher gehalten werden. Dies entspricht also bei dem aktuellen Beispiel dem Block auf dem die i-node der Datei gespeichert ist und der Block auf dem die single indirect Blöcke gespeichert sind. Wenn die Dateien allerdings weniger als 12 KiB groß sind, muss für eine Datei nur die entsprechende i-node in den Speicher geladen werden. Bei FAT muss im Gegensatz immer die komplette File Allocation Tabelle (hier 48 MiB) in den Speicher geladen werden.

Aufgabe 4 ext Inodes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mode		uid		size				atime				ctime			
mtime				dtime				gid		links_count		blocks			
flags				version											
direct_blocks[12]															
								single_indirect_blocks				double_indirect_blocks			
triple_indirect_blocks				generation				xattr_addr				size_high			
faddr				frag	fsize	*reserved*		uid_high		gid_high		*reserved*			

Die Felder geben dabei an:

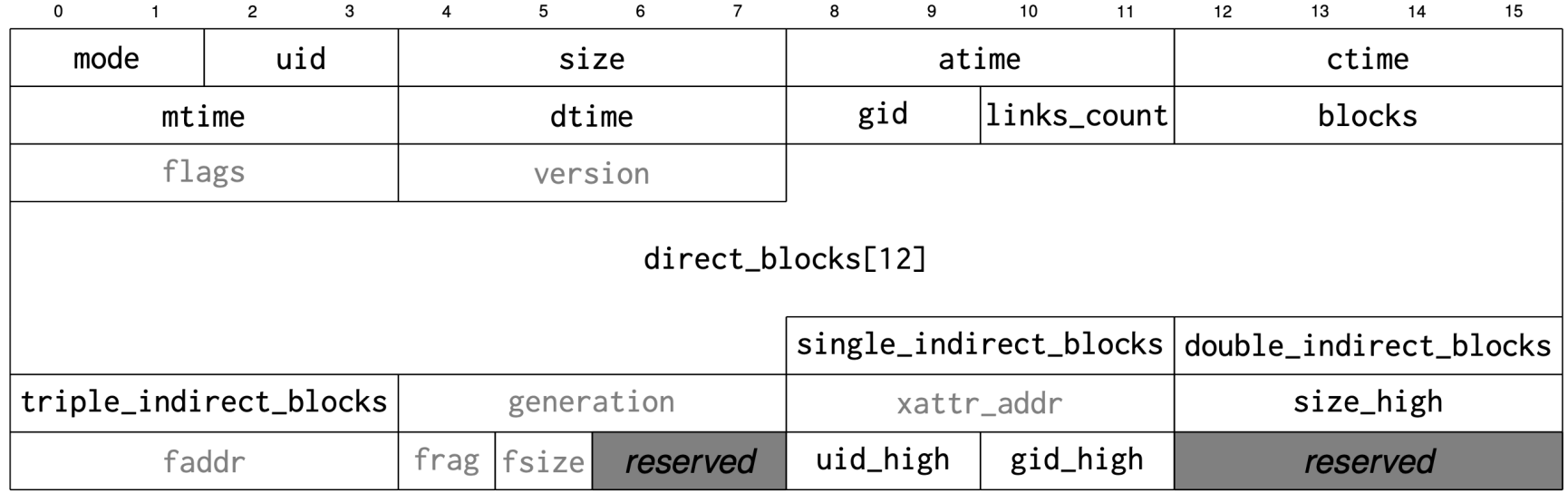
- `mode`: Art der Datei (reguläre Datei: 8, Directory: 4, symbolic Link: 10, named Pipe: 1 etc.)
  + Rechte: `r` (read), `w` (write) und `x` (execute) je für den Besitzer, die Nutzer der Gruppe und alle anderen

15				8	7						0
Typ	SUID	SGID	sticky	user read	user write	user exec	group read	group write	group exec	other read	other write	other exec

  Bei Verzeichnissen haben die Rechte-Bits eine andere Bedeutung: `r`: Auflisten der Dateien; `w`: Hinzufügen/Umbenennen/Entfernen von Dateien; `x`: Nutzen der Dateien im Verzeichnis
- `uid`, `gid`: Nutzer-ID des Besitzers, Gruppen-ID
- `size`: Größe der Datei in Bytes
- `atime`, `ctime`, `mtime`, `dtime`: Access-, Change-, Modification- und Deletion-Time. `mtime` gibt an, wann zuletzt die Daten geändert wurden; `ctime` gibt an, wann zuletzt die Inode geändert wurde (außer `atime`).
- `links_count`: Anzahl der Hardlinks auf Inode
- `blocks`: Anzahl der allozierten 512-Byte Blöcke[2]

Alle Werte sind in little-Endian gespeichert. Mit `stat` können sie für eine Datei ausgegeben werden.

Aufgabe 4 ext Inodes



a) Gegeben sei folgender Output von `ls -la`. Zu welchen Feldern der Inode korrespondieren die Spalten jeweils?

```
total 150K
drwxr-xr-x  3 josef josef     1024 Dec 31 16:36 .
drwxr-xr-x 27 root  root      1024 Dec 31 16:38 ..
-rw-r--r--  2 josef josef       13 Dec 31 15:54 file.txt
-rw-rw----  1 josef gbs_uel 145013 Dec 31 15:57 gbs-endterm-solution-ws2022.pdf
-rwx------  1 josef josef    14866 Dec 31 16:37 generate_valid_itsec_flags.sh
drwxr-xr-x  5 josef josef     3072 Dec 31 16:39 Memes
```

Aufgabe 4 ext Inodes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mode		uid		size				atime				ctime			
mtime				dtime				gid		links_count		blocks			
flags				version											
direct_blocks[12]															
								single_indirect_blocks				double_indirect_blocks			
triple_indirect_blocks				generation				xattr_addr				size_high			
faddr				frag	fsize	*reserved*		uid_high		gid_high		*reserved*			

```
total 150K
drwxr-xr-x   3 josef josef      1024 Dec 31 16:36 .
drwxr-xr-x  27 root  root       1024 Dec 31 16:38 ..
-rw-r--r--   2 josef josef        13 Dec 31 15:54 file.txt
-rw-rw----   1 josef gbs_uel 145013 Dec 31 15:57 gbs-endterm-solution-ws2022.pdf
-rwx------   1 josef josef     14866 Dec 31 16:37 generate_valid_itsec_flags.sh
drwxr-xr-x   5 josef josef      3072 Dec 31 16:39 Memes
```

- Änderungsdatum (`mtime`)
- Dateigröße
- Gruppe
- Besitzer
- (Hard) Link Count
- Mode: Rechte (Owner, Group, other)
- Mode: Typ

Aufgabe 4 ext Inodes

b) Wer könnte in die Datei `file.txt` schreiben?

```
total 150K
drwxr-xr-x  3 josef josef     1024 Dec 31 16:36 .
drwxr-xr-x 27 root  root      1024 Dec 31 16:38 ..
-rw-r--r--  2 josef josef       13 Dec 31 15:54 file.txt
-rw-rw----  1 josef gbs_uel 145013 Dec 31 15:57 gbs-endterm-solution-ws2022.pdf
-rwx------  1 josef josef    14866 Dec 31 16:37 generate_valid_itsec_flags.sh
drwxr-xr-x  5 josef josef     3072 Dec 31 16:39 Memes
```

Aufgabe 4 ext Inodes

b) Wer könnte in die Datei `file.txt` schreiben?

```
total 150K
drwxr-xr-x  3 josef josef     1024 Dec 31 16:36 .
drwxr-xr-x 27 root  root      1024 Dec 31 16:38 ..
-rw-r--r--  2 josef josef       13 Dec 31 15:54 file.txt
-rw-rw----  1 josef gbs_uel 145013 Dec 31 15:57 gbs-endterm-solution-ws2022.pdf
-rwx------  1 josef josef    14866 Dec 31 16:37 generate_valid_itsec_flags.sh
drwxr-xr-x  5 josef josef     3072 Dec 31 16:39 Memes
```

Der Besitzer der Datei ist `josef`. Für ihn sind die Rechte `rw-`, d.h. er kann die Datei lesen und beschreiben.
Die Gruppe der Datei ist `josef`. Für diese sind die Rechte allerdings nur `r--`, die Mitglieder der Gruppe können die Datei also nur lesen (Benutzer `josef` selbst ausgenommen).
Alle anderen haben die Rechte `r--`, auch sie können die Datei daher nur lesen.

Wichtig ist dabei, zusätzlich zu beachten, ob die verschiedenen Nutzer überhaupt auf die Dateien der Verzeichnisse zugreifen dürfen: Sowohl `.` als auch `..` haben jeweils das `x`-Bit für alle Nutzer gesetzt, somit dürfen sie.
Über `../..`, `../../..` usw. haben wir hier keine Information.

Aufgabe 4 ext Inodes

c) Worin besteht der Unterschied zwischen sym- und hardlinks?

Aufgabe 4 ext Inodes

c) Worin besteht der Unterschied zwischen sym- und hardlinks?

Bei Hardlinks verweisen mehrere Einträge in (ggf. verschiedenen) Verzeichnissen gleichermaßen auf eine gemeinsame Inode. Damit sind neben den Nutzdaten auch die ganzen Metadaten (Rechte, Zeiten, Besitzer, etc.) geteilt. Wird nun einer der beiden Einträge in der Verzeichnishierarchie entfernt, so wird in der Inode zwar der Link-Counter dekrementiert, die Daten sind aber über den anderen Eintrag noch erreichbar. Erst wenn der letzte Link entfernt wurde (d.h. sobald der Link-Counter gleich 0 ist), kann die Inode samt den Daten der Datei gelöscht werden.

Ein Symlink hingegen verweist nicht auf eine Inode, sondern beinhaltet lediglich einen Pfad auf eine weitere Datei. Wird der Link nun geöffnet, so liest der Kernel den Pfad aus und öffnet stattdessen die entsprechende Datei, auf die verwiesen wird. Sobald die Zieldatei aber gelöscht/verschoben wird, zeigt der Link auf eine nicht-existente Datei und ist somit invalide.

Symlinks verändern daher auch nicht den Link-Count der Inode der Zieldatei.

Aufgabe 4 ext Inodes

d) Wo steht der Dateiname?

Aufgabe 4 ext Inodes

d) Wo steht der Dateiname?

Der Dateiname steht im Verzeichnis. Damit können verschiedene Hardlinks verschiedene Namen haben, obwohl sie sich eine Inode teilen.

Außerdem ist der lookup einer Datei im Verzeichnis somit effizienter, da alle Dateinamen in den Blöcken des Verzeichnis selbst stehen und nicht alle Inodes der Dateien des Verzeichnis (welche quer über die Festplatte verteilt sein könnten) gelesen werden müssen.