Marek Kubica, kubica@in.tum.de

Punkteverteilung:

1	2	3	Σ

Aufgabe (1)

(a) Die Blöcke sind 1KB groß, in einem Inode sind 10 Blöcke referenziert, also kann eine Datei maximal 10KB groß sein. An sich sind im Block mit dem Inode aber noch (1024 - 10 * (32 / 8)) Byte frei, die 984 / 4 = 246 weitere Adressen zu anderen Blöcken aufnehmen könnten.

Irgendwie ist die Aufgabenstellung noch dümmer als ich, denn diese Antwort ist trivial und wie die 32-Bit Adresse relevant ist wird nicht ganz klar.

Es gäbe die Möglichkeit eine interessantere Antwort als 10KB zu geben, wenn die Aufgabenstellung etwas von (ggf. mehrfach) indirekten Blöcken sagen würde. So kann nur gemutmaßt werden, ob nach den 10 Blöcken noch ein 11, einfach indirekter Block ist. Oder ganz viele (Platz dafür gäbe es im Block ja zu genüge). Alternativ könnten auch mehrfach indirekt referenzierte Blöcke hinter den direkten Blöcken stehen die dann auch quasi beliebig oft indirekt sein können (Tanenbaum stellt bis zu dreifach vor, aber natürlich geht auch wesentlich mehr). Somit existieren für diese Aufgabestellung quasi beliebige Anzahlen von Antworten die richtig sein können, wenn eine entsprechende Kombination von indirekten Blöcken vorliegt.

Da auch nicht bekannt ist von was für einem Unix-Dateisystem die Rede ist, kann man auch schlecht im Internet nachschauen, eigentlich nur raten.

Kris Köhntopp (`http://kris.koehntopp.de/artikel/unix/dateisysteme/node6.html`) beschreibt das System V-Dateisystem so dass ein Inode 64 KB groß ist, somit wären nach den 10 direkten Adressen noch Platz für 6 (mehrfach) indirekte Blöcke. Davon werden 3 benutzt: der 11. ist einfach indirekt, bietet also 265 weitere Adressen, also erhöht die maximale Dateigröße um 265 KB. Der 12. ist zweifach indirekt, bietet also 256 * 256 = 65536 neue Adressen, und erhöht somit die maximale Dateigröße um 65536 KB. Der 13. Block ist dreifach indirekt, also 256 * 256 * 256 neue Adressen, 16777216 KB weiterer Speicher. Insgesamt also 10 + 256 + +65536 + 16777216 = 16843018 KB also etwas mehr als 16 GB.

(b)
a) Die Datei gehört mir, da beim kopieren die Berechtigungen nicht geändert werden. Wäre auch gar nicht möglich da auf Unix-Systemen ein User nicht einfach eine Datei äufgeben"kann, d.h. einem anderem User übergeben kann.
b) Der Kommilitone kann sich die Datei mit ls anzeigen mir aber nicht, denn obwohl ich mit dem x-Bit des Verzeichnisses theoretisch das Verzeichnis durchsuchen könnte besitze ich nicht das passende r-Bit um den Ordner zu lesen.
c) Ich kann die Datei lesen denn ich habe das Owner-r-Bit an der Datei und mein Kommilitone nicht.
d) Ich kann die Datei verändern, da ich das Owner-w-Bit habe, im Gegensatz zu meinen Kommilitonen. Selbiges wie oben.
e) Beide können die Datei löschen, da beide das Verzeichnis modifizieren dürfen (über Owner-w-Bit und Group/Other-w-Bit).

Aufgabe (2)

(a) 250 KB, das entspricht 250 nötigen Blöcken. 10 Block sind direkt im Inode verlinkt, damit bleiben noch 240 nötige Blöcke. Im indirekten Verweis können 256 Adressen

für Blöcke untergebracht werden, somit kann die Datei nun vollständig auf die Festplatte ausgeschrieben werden. Der Verwaltungsaufwand für diese Datei beläuft sich auf 2 Blöcke: den Inode und den indirekten Verweis.

(b) 16 GB das sind 16777216 KB, was 16777216 Blöcken entspricht. Um diese Blöcke zu adressieren benötigt man 2^{24} Bit. Die FAT enthält also für jeden Block 24 Bit, somit ist die größe 16777216 * 24 Bit was gleich 402653184 Byte ist. Umgerechnet also 48 MB.

(c) Inodes sind in der Situation vorteilhafter, weil für jede geöffnete Datei nur der Inode in den Speicher geladen werden muss. Dieser ist wesentlich kleiner als die FAT.

Aufgabe (3)

NTFS ist das Standarddateisystem von Windows NT und als solches eine wesentliche Verbesserung gegenüber FAT (was mit daran liegt, dass Microsoft kompetente Leute wie Dave Cutler eingesetzt hat, das Dateisystem zu entwerfen).

Die Länge der Dateinamen ist 255 Zeichen was auf den ersten Blick vergleichbar ist mit den 255 Zeichen die moderne Unix-Dateisysteme anbieten, aber in Unix sind es 255 Byte in NTFS hingegen UTF-16 Code Units, d.h. 255 * 16 Byte - UTF-16 kodiert jedes Zeichen aus der BMP (in der sowieso die meisten in der Realität vorkommenden Zeichen vertreten sind) in 16 Byte, für weitere nutzt es ein weiteres 16-Bit Wort. Somit kann man vereinfachend sagen dass Dateinamen in NTFS 255 Unicode-Zeichen enthalten können und Unix 255 ASCII-Zeichen (üblicherweise wird UTF-8 verwendet, womit ein Zeichen zwischen 1 und 4 Byte groß werden kann).

Neben der für 1993 mutigen Entscheidung auf Unicode zu setzen bietet NTFS für jede Datei mehrere Attribute die Byteströme enthalten können. Es gibt ein unbenanntes Attribut mit dem Dateiinhalt (analog zu Unix) aber auch weitere Attribute wie den Dateinamen und die Objekt-ID. Das Konzept erinnert an die Ressource-Forks aus früheren Mac OS-Versionen (vor OS X) ist aber ein Stück flexibler. Damit ist es möglich sämtliche Metainformationen die das OS aber auch Userprogramme benötigen können, direkt in die Datei zu hängen. Ein Beispiel dafür sind ACLs die direkt an der Datei hängen können ohne dass man an der Struktur des Dateisystems etwas ändern müsste. Unter Unix ist dies in viele Dateisysteme erst nachträglich integriert worden und teilweise eine mount-Option. Eine interessante Übersicht bietet `http://en.wikipedia.org/wiki/Extended_file_attributes`.

Was Blockgrößen angeht unterstützt NTFS Cluster (der Microsoft-Name für Blöcke) von 512 Byte bis 64 KB wobei 4KB die populärste Wahl ist. ext3 unter Linux unterstützt Blockgrößen von 1KB bis 8KB; 2KB war auf ext3 früher Standard, wurde aber im Hinblick auf ext4 auf 4KB angehoben. 8KB wird nur auf Architekturen unterstützt die 8KB Paging unterstützen.

Die Master File Table ist das grobe äquivalent zum Superblock auf Unix-Dateisystemen. Sie enthält Informationen zu jeder Datei auf dem Dateisystem und ist in 1KB Segmente geteilt.Interessant ist in diesem Zusammenhang dass die MFT eine normale Datei ist, die im Dateisystem liegt.

Rechte wurden bis NT 4.0 in den MFT-Einträgen der betreffenenden Dateien verwaltet, ab Windows 2000 wurden die Rechteangaben in eine eigene Dati geschrieben, so dass sich mehrere Dateien die gleichen Rechte teilen konnten.