Karlsruhe Institute of Technology

Operating Systems 2016/17
Solutions for Assignment 13

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

T-Question 13.1: File System Implementation

a. What three kernel data structures are required to manage open files? Explain the meaning of each. **3 T-pt**

Solution:

- ***Open File State****: For each open file in the system, the kernel allocates a data structure to hold the state of the open file. This structures includes the current seek-pointer, granted access (read-only, etc.), and others. The state objects are stored in a system-wide open file-table.*
- ***Handle Mapping Table****: To identify an open file, the user process must somehow hold a reference to the it. Using pointers into kernel memory is severely insecure, because the user could provide false pointers and thus lead the kernel to interpret arbitrary memory as open file state objects (probably crashing the system). In addition, exposing pointers to kernel data structures to user-mode is considered an information leak.*

  A per-process open file table is thus used as indirection. The table is stored in the process control block (PCB) in kernel memory and user-mode code only provides indices into this (kernel-controlled) table to translate a handle to a kernel pointer (for a state object).

  In Linux, a file handle is called file descriptor.
- ***File Control Block****: Each file in the file system needs to be described by some form of data structure. In UNIX operating systems files are represented through inodes.*

  To manage open files, each open file state object must also hold a reference to the actual file and thus contains a reference to the corresponding file control block (inode) structure (this can be a pointer if the OS loads FCBs into memory for open files).

  Note that if additional layers are added (e.g., a VFS), the open file state object might only point to intermediate data structures (e.g., vNode).

b. Why is the file name not stored in the inode? **1 T-pt**

Solution:
Although an inode identifies a single file, there might be multiple hard-links to the same file (i.e., the file has more than one name). If the inode would contain the file name it would not be clear what name to choose for a certain directory.

Another benefit of not storing the file name in the inode, but in the directory is the possibility to increase spatial locality for directory listings.

c. How does the file system determine if an inode and thus the blocks allocated to the file can be deleted? **1 T-pt**

Solution:
The file system keeps a hard-link counter, which stores the number of references to the inode. If the counter reaches zero, there are no more references to the file in the directory tree, and the file system can delete the inode and the associated blocks.

d. What is the benefit of a file allocation table (FAT) compared to chained allocation? **1 T-pt**

Solution:
Both methods use linking to determine the sequence of blocks that make up a file. However, in contrast to chained allocation, a file allocation table separates the links from the data blocks and thereby improves spatial locality for the linking information. The FAT therefore can often be (partially) loaded into RAM, which improves access performance.

e. Explain the concept of indexed disk space allocation. What approach allows this allocation type to represent very large files while still being efficient for small files? **2 T-pt**

Solution:
Files may be comprised of an arbitrary set of blocks. To identify the blocks that belong to a file, indexed space allocation stores a list of block pointers per file. The list is ordered in the right way to reflect the file's contents.

To address large files indexed space allocation uses multiple levels of index blocks, comparable to a page table hierarchy. This structure introduces indirections that harm performance. To maintain a good performance for small files, one often combines multiple such tables with increasing levels of indirection with increasing file size. The first pointers in the inode thereby directly point to data blocks (good for small files), while pointers at the end of the inode point to other index blocks and hierarchies of index blocks.

Total: 8 T-pt