COMP3231/9201/3891/9283 Operating Systems 2020/T1

UNSW

Tutorial Week 7

Questions

Files and file systems

1. Consider a file currently consisting of 100 records of 400 bytes. The filesystem uses *fixed blocking*, i.e. one 400 byte record is stored per 512 byte block. Assume that the file control block (and the index block, in the case of indexed allocation) is already in memory. Calculate how many disk I/O operations are required for contiguous, linked, and indexed (single-level) allocation strategies, if, for one record, the following conditions hold. In the contiguous-allocation case, assume that there is no room to grow at the beginning, but there is room to grow at the end of the file. Assume that the record information to be added is stored in memory.

    a. The record is added at the beginning.
    b. The record is added in the middle.
    c. The record is added at the end.
    d. The record is removed from the beginning.
    e. The record is removed from the middle.
    f. The record is removed from the end.

---

2. Old versions of UNIX allowed you to write to directories. Newer ones do not even allow the superuser to write to them? Why? Note that many unices allow you read directories.

---

3. Why is there VFS Layer in Unix?

---

4. How does choice of block size affect file system performance. You should consider both sequential and random access.

---

5. Why does Linux pre-allocate up to 8 blocks on a write to a file.

---

6. Linux uses a *buffer cache* to improve performance. What is the drawback of such a cache? In what scenario is it problematic? What alternative would be more appropriate where a buffer cache is inappropriate?

---

7. What is the structure of the contents of a directory? Does it contain attributes such as creation times of files? If not, where might this information be stored?

---

8. The Unix inode structure contains a reference count. What is the reference count for? Why can't we just remove the inode without checking the reference count when a file is deleted?

---

9. Inode-based filesystems typically divide a file system partition into *block groups*. Each block group consists of a number of contiguous physical disk blocks. Inodes for a given block group are stored in the same physical location as the block groups. What are the advantages of this scheme? Are they any disadvantages?

---

10. Assume an inode with 10 direct blocks, as well as single, double and triple indirect block pointers. Taking into account creation and accounting of the indirect blocks themselves, what is the largest possible number of block reads and writes in order to:

    a. Read 1 byte
    b. Write 1 byte

    Assume the inode is cached in memory.

---

11. Assume you have an inode-based filesystem. The filesystem has 512 byte blocks. Each inode has 10 direct, 1 single indirect, 1 double indirect, and 1 triple indirect block pointer. Block pointers are 4 bytes each. Assume the inode and any block free list is always in memory. Blocks are not cached.

    a. What is the maximum file size that can be stored before
        1. the single indirect pointer is needed?
        2. the double indirect pointer is needed?
        3. the triple indirect pointer is needed?
    b. What is the maximum file size supported?
    c. What is the number of disk block reads required to read 1 byte from a file
        1. in the best case?
        2. in the worst case?
    d. What is the number of disk block reads and writes required to write 1 byte to a file
        1. in the best case?
        2. in the worst case?

---

12. A typical UNIX inode stores both the file's size and the number of blocks currently used to store the file. Why store both? Should not blocks = size / block size?

---

13. How can deleting a file leave a inode-based file system (like ext2fs in Linux) inconsistent in the presence of a power failure.

---

14. How does adding journalling to a file system avoid corruption in the presence of unexpected power failures.

Page last modified: 12:26pm on Monday, 1st of April, 2019

Screen Version