COMP3231/9201/3891/9283 Operating Systems 2021/T1

Tutorial Week 5

Questions and Answers

Memory Hierarchy and Caching

1. Describe the memory hierarchy. What types of memory appear in it? What are the characteristics of the memory as one moves through the hierarchy? How can do memory hierarchies provide both fast access times and large capacity?

    The memory hierarchy is a hierarchy of memory types composed such that if data is not accessible at the top of the hierarchy, lower levels of the hierarchy are accessed until the data is found, upon which a copy (usually) of the data is moved up the hierarchy for access.

    Registers, cache, main memory, magnetic disk, CDROM, tape are all types of memory that can be composed to form a memory hierarchy.

    In going from the top of the hierarchy to the bottom, the memory types feature decreasing cost per bit, increasing capacity, but also increasing access time.

    As we move down the hierarchy, data is accessed less frequently, i.e. frequently accessed data is at the top of the hierarchy. The phenomenon is called "locality" of access, most accesses are to a small subset of all data.

2. Given that disks can stream data quite fast (1 block in tens of microseconds), why are average access times for a block in milliseconds?

    Seek times are in milliseconds (e.g. .5 millisecond track to track, 8 millisecond inside to outside), and rotational latency (1/2 rotation) is in milliseconds (e.g. 2 milliseconds for 15,000rpm disk).

3. You have a choice of buying a 3 Ghz processor with 512K cache, and a 2 GHz processor (of the same type) with a 3 MB cache for the same price. Assuming memory is the same speed in both machines and is much less than 2GHz (say 400MHz). Which would you purchase and why? Hint: You should consider what applications you expect to run on the machine.

    If you are only running an small application (or a large one, that accesses only a small subset), then the 3GHz processor will be much faster. If you are running a large application access a larger amount of memory than 512K but generally less than 3MB, the 2GHz processor should be faster as the 3 GHz processor will be limited by memory speed.

Files and file systems

4. 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.

	Contiguous	Linked	Indexed
a.	100r/101w	0r/1w	0r/1w
b.	50r/51w	50r/2w	0r/1w
c.	0r/1w	100r/2w	0r/1w
d.	0r/0w	1r/0w	0r/0w
e.	49r/49w	50r/1w 51r/1w	0r/0w
f.	0r/0w	99r/1w	0r/0w

---

5. In the previous example, only 400 bytes is stored in each 512 byte block. Is this wasted space due to internal or external fragmentation?

   Internal fragmentation

---

6. 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.

   To prevent total corruption of the fs. eg `cat /dev/zero > /`

---

7. Given a file which varies in size from 4KiB to 4MiB, which of the three allocation schemes (*contiguous, linked-list, or i-node based*) would be suitable to store such a file? If the file is access randomly, how would that influence the suitability of the three schemes?

   Contiguous is not really suitable for a variable size file as it would require 4MiB to be pre-allocated, which would waste a lot of space if the file is generally mush smaller. Either linked-list of i-node-based allocation would be preferred. Adding random access to the situation (supported well by contiguous or i-node based), which further motivate i-node-based allocation to be the most appropriate.

---

8. Why is there VFS Layer in Unix?

   - It provides a framework to support multiple file system types concurrently without requiring each file system to be aware of other file system types.
   - Provides transparent access to all supported file systems including network file systems (e.g. NFS, CODA)
   - It provides a clean interface between the file system independent kernel code and the file system specific kernel code.
   - Provide support for *special* file system types like `/proc`.

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

    - Sequential Access

      The larger the block size, the fewer I/O operations required and the more contiguous the disk accesses. Compare loading a single 16K block with loading 32 512-byte blocks.

    - Random Access

      The larger the block size, the more unrelated data loaded. Spatial locality of access can improve the situation.

---

10. Is the `open()` system call in UNIX essential? What would be the consequence of not having it?

    It is not absolutely essential. The `read` and `write` system calls would have to be modified such that:

    - The filename is passed in on each call to identify the file to operate on
    - With a file descriptor to identify the open session that is returned by `open`, the sycalls would also need to specify the offset into the file that the syscall would need to use.
    - Effectively opening and closing the file on each `read` or `write` would reduce performance.

---

11. Some operating system provide a *rename* system call to give a file a new name. What would be different compared to the approach of simply copying the file to a new name and then deleting the original file?

    The rename system call would just change the string of characters stored in the directory entry. A copy operation would result in a new directory entry, and (more importantly) much more I/O as each block of the original file is copied into a newly allocated block in the new file. Additionally, the original file blocks need de-allocating after the copy finishes, and the original name removed from the directory. A rename is much less work, and thus way more efficient than the copy approach.

---

12. In both UNIX and Windows, random file access is performed by having a special system call that moves the *current position* in the file so the subsequent `read` or `write` is performed from the new position. What would be the consequence of not having such a call. How could random access be supported by alternative means?

    Without being able to move the file pointer, random access is either extremely inefficient as one would have to read sequentially from the start each time until the appropriate offset is arrived at, or the an extra argument would need to be added to `read` or `write` to specify the offset for each operation.

Page last modified: 9:13am on Tuesday, 9th of February, 2021

Screen Version