Tutorial Week 5

CPU registers
Memory
Cache
Disk
Disk Tape

Questions

Memory Hierarchy and Caching

the characteristics of the memory as one moves through the hierarchy is that you have larger storage but the speed decreases exponentially

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?

to provide both fast access times and large capacity, we need to use cache

---

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?

because the data on the disk are not continuous, you have data at different locations on the disk, so to access a block of data, you are limited by the physical constraints such as spinning speed, how fast the arm can go in and out. therefore the average access times for a block in much slower compared to the speed of streaming data

---

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.

Files and file systems

internal fragmentation

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	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?

---

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

---

if everything is fixed size, then contiguous can also be used

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?

linked list and i-node based would be suitable to store such a file. If the file is access randomly, i-node would be the best way to store such a file

---

8. Why is there VFS Layer in Unix?

because the OS has to deal with different devices with different underlying file systems, such as fat, ext2, ext3, ntfs etc. Having a VFS provides the interface to support multiple file systems.

1. It provides a framework to support multiple file system types concurrently without requiring each file system to be aware of other file system types.
2. Provides transparent access to all supported file systems including network file systems (e.g. NFS, CODA)
3. It provides a clean interface between the file system independent kernel code and the file system specific kernel code.
4. 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.

   depending on the allocation schemes, if we have a contiguous fs, block size doesn't affect the performance because it reads from beginning to the end of the file. It has very simple implementation and good performance for sequential access. If we have a linked list fs, the

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 syscalls would have to be modified such that :1. the filename is passed in on each call to identify the file to operate on. 2. with a fd to identify the open session that is returned by open, the syscall would also need to specify the offset into the file that the syscall would need to use 3. 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 syscall would just change the string of characters stored in the directory entry. A copy operation would result in a new directory entry, and 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 less way more efficient

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?

    if not having such special system call that moves the current position, then we always traverse the file from start to end, which would become less efficient and time consuming to find the correct offset in the file. Random access would need extra argument to specify the offset for each position.