CSCI 333
Data Structures

Chapter 8
18, 21, and 23 October 2002

Notes with the dark blue background were prepared by the textbook author

Clifford A. Shaffer
Department of Computer Science
Virginia Tech

Primary vs. Secondary Storage

Primary storage: Main memory (RAM)

Secondary Storage: Peripheral devices
- Disk drives
- Tape drives

Comparisons

Medium	Early 1996	Mid 1997	Early 2000	Fall 2002
RAM	$45.00	7.00	1.50	0.500
Disk	0.25	0.10	0.01	.003
Floppy	0.50	0.36	0.25	*who*
Tape	0.03	0.01	0.001	*cares*

RAM is usually volatile.

RAM is about 1/4 million times faster than disk.

Golden Rule of File Processing

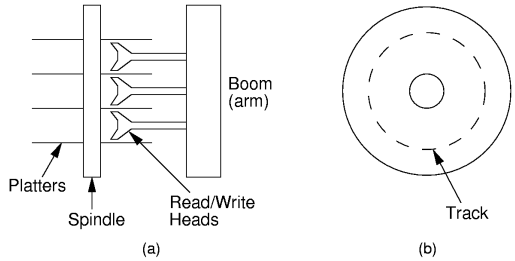
Minimize the number of disk accesses!

1. Arrange information so that you get what you want with few disk accesses.
2. Arrange information to minimize future disk accesses.

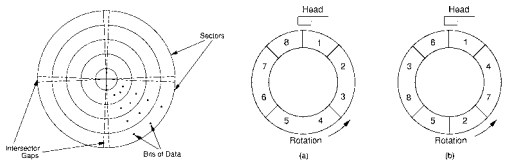
An organization for data on disk is often called a file structure.

Disk-based space/time tradeoff: Compress information to save processing time by reducing disk accesses.

Disk Drives



Sectors



A sector is the basic unit of I/O.

Interleaving factor: Physical distance between logically adjacent sectors on a track.

Zoned disks

Zones are composed of tracks

Zones at outer edge of disks have

More sectors

Higher data transfer rate

Logical Blocks

- Address by logical blocks number
  - Not by C:H:S
    - Cylinder, Head, Sector
- Used in most modern disks
- Makes life easier for operating system
  - No C:H:S calculation
  - Controller can do bad-block remapping

Terms

Locality of Reference: When record is read from disk, next request is likely to come from near the same place in the file.

Cluster: Smallest unit of file allocation, usually several sectors.

Extent: A group of physically contiguous clusters.

Internal fragmentation: Wasted space within sector if record size does not match sector size; wasted space within cluster if file size is not a multiple of cluster size.

Seek Time

Seek time: Time for I/O head to reach desired track. Largely determined by distance between I/O head and desired track.

Track-to-track time: Minimum time to move from one track to an adjacent track.

Average Access time: Average time to reach a track for random access.

Other Factors

Rotational Delay or Latency: Time for data to rotate under I/O head.

- One half of a rotation on average.
- At 7200 rpm, this is 8.3/2 = 4.2ms.

Transfer time: Time for data to move under the I/O head.

- At 7200 rpm: Number of sectors read/Number of sectors per track * 8.3ms.

Disk Spec Example

16.8 GB disk on 10 platters = 1.68GB/platter
13,085 tracks/platter
256 sectors/track
512 bytes/sector
Track-to-track seek time: 2.2 ms
Average seek time: 9.5ms
4KB clusters, 32 clusters/track.
Interleaving factor of 3.
5400RPM

Disk Access Cost Example (1)

Read a 1MB file divided into 2048 records of 512 bytes (1 sector) each.

Assume all records are on 8 contiguous tracks.

First track: 9.5 + 11.1/2 + 3 x 11.1 = 48.4 ms

Remaining 7 tracks: 2.2 + 11.1/2 + 3 x 11.1 = 41.1 ms.

Total: 48.4 + 7 * 41.1 = 335.7ms

Disk Access Cost Example (2)

Read a 1MB file divided into 2048 records of 512 bytes (1 sector) each.

Assume all file clusters are randomly spread across the disk.

256 clusters. Cluster read time is (3 x 8)/256 of a rotation for about 1 ms.

256(9.5 + 11.1/2 + (3 x 8)/256) is about 38877 ms. or nearly 40 seconds.

How Much to Read?

Read time for one track:
9.5 + 11.1/2 + 3 x 11.1 = 48.4ms.

Read time for one sector:
9.5 + 11.1/2 + (1/256)11.1 = 15.1ms.

Read time for one byte:
9.5 + 11.1/2 = 15.05 ms.

Nearly all disk drives read/write one sector at every I/O access.
- Also referred to as a page.

More terminology

- Drive interface standard
  - ATA
  - SCSI
  - RAID
- Device access method
  - DMA, UDMA

Some disk specifications

- IBM Travelstar
- IBM Ultrastar
- IBM Enterprise Storage Server
- IBM McDATA Intrepid 6140 Director
- IBM Microdrive

Buffers

The information in a sector is stored in a buffer or cache.

If the next I/O access is to the same buffer, then no need to go to disk.

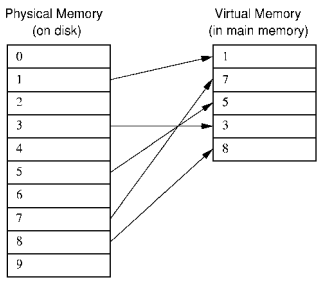
There are usually one or more input buffers and one or more output buffers.

Buffer Pools

A series of buffers used by an application to cache disk data is called a buffer pool.

Virtual memory uses a buffer pool to imitate greater RAM memory by actually storing information on disk and "swapping" between disk and RAM.

Buffer Pools



Organizing Buffer Pools

Which buffer should be replaced when new data must be read?

First-in, First-out: Use the first one on the queue.

Least Frequently Used (LFU): Count buffer accesses, reuse the least used.

Least Recently used (LRU): Keep buffers on a linked list. When buffer is accessed, bring it to front. Reuse the one at end.

Design Issues

Disadvantage of message passing:

- Messages are copied and passed back and forth.

Disadvantages of buffer passing:

- The user is given access to system memory (the buffer itself)
- The user must explicitly tell the buffer pool when buffer contents have been modified, so that modified data can be rewritten to disk when the buffer is flushed.
- The pointer might become stale when the bufferpool replaces the contents of a buffer.

External Sorting

Problem: Sorting data sets too large to fit into main memory.

- Assume data are stored on disk drive.

To sort, portions of the data must be brought into main memory, processed, and returned to disk.

An external sort should minimize disk accesses.

Model of External Computation

Secondary memory is divided into equal-sized blocks (512, 1024, etc…)

A basic I/O operation transfers the contents of one disk block to/from main memory.

Under certain circumstances, reading blocks of a file in sequential order is more efficient. (When?)

Primary goal is to minimize I/O operations.

Assume only one disk drive is available.

Key Sorting

Often, records are large, keys are small.
- Ex: Payroll entries keyed on ID number

Approach 1: Read in entire records, sort them, then write them out again.

Approach 2: Read only the key values, store with each key the location on disk of its associated record.

After keys are sorted the records can be read and rewritten in sorted order.

Simple External Mergesort (1)

Quicksort requires random access to the entire set of records.

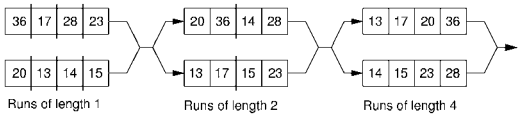
Better: Modified Mergesort algorithm.
- Process n elements in $\Theta(\log n)$ passes.

A group of sorted records is called a *run*.

Simple External Mergesort (2)

- Split the file into two files.
- Read in a block from each file.
- Take first record from each block, output them in sorted order.
- Take next record from each block, output them to a second file in sorted order.
- Repeat until finished, alternating between output files. Read new input blocks as needed.
- Repeat steps 2-5, except this time input files have runs of two sorted records that are merged together.
- Each pass through the files provides larger runs.

Simple External Mergesort (3)



Problems with Simple Mergesort

Is each pass through input and output files sequential?

What happens if all work is done on a single disk drive?

How can we reduce the number of Mergesort passes?

In general, external sorting consists of two phases:
- Break the files into initial runs
- Merge the runs together into a single run.

Breaking a File into Runs

General approach:

- Read as much of the file into memory as possible.
- Perform an in-memory sort.
- Output this group of records as a single run.

Replacement Selection (1)

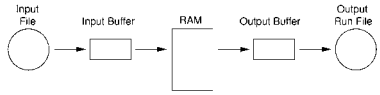
- Break available memory into an array for the heap, an input buffer, and an output buffer.
- Fill the array from disk.
- Make a min-heap.
- Send the smallest value (root) to the output buffer.

Replacement Selection (2)

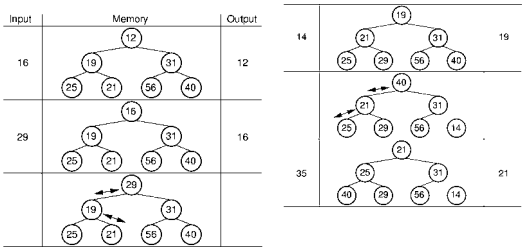
- If the next key in the file is greater than the last value output, then
  - Replace the root with this key

  else
  - Replace the root with the last key in the array

  Add the next record in the file to a new heap (actually, stick it at the end of the array).



RS Example



Snowplow Analogy (1)

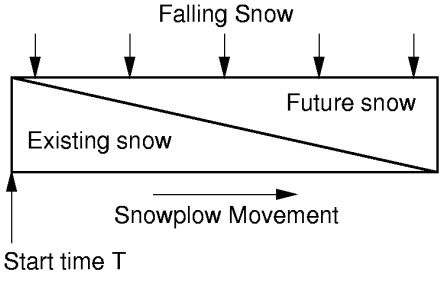
Imagine a snowplow moving around a circular track on which snow falls at a steady rate.

At any instant, there is a certain amount of snow *S* on the track. Some falling snow comes in front of the plow, some behind.

During the next revolution of the plow, all of this is removed, plus 1/2 of what falls during that revolution.

Thus, the plow removes 2*S* amount of snow.

Snowplow Analogy (2)



Problems with Simple Merge

Simple mergesort: Place runs into two files.

- Merge the first two runs to output file, then next two runs, etc.

Repeat process until only one run remains.

- How many passes for r initial runs?

Is there benefit from sequential reading?

Is working memory well used?

Need a way to reduce the number of passes.

Multiway Merge (1)

With replacement selection, each initial run is several blocks long.

Assume each run is placed in separate file.

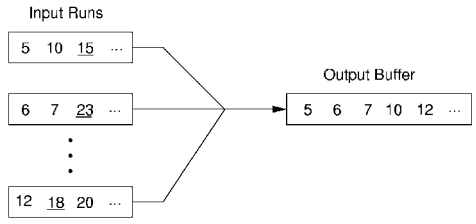
Read the first block from each file into memory and perform an *r*-way merge.

When a buffer becomes empty, read a block from the appropriate run file.

Each record is read only once from disk during the merge process.

Multiway Merge (2)

In practice, use only one file and seek to appropriate block.



Limits to Multiway Merge (1)

Assume working memory is *b* blocks in size.

How many runs can be processed at one time?

The runs are 2*b* blocks long (on average).

How big a file can be merged in one pass?

Limits to Multiway Merge (2)

Larger files will need more passes -- but the run size grows quickly!

This approach trades (log b) (possibly) sequential passes for a single or very few random (block) access passes.

General Principles

A good external sorting algorithm will seek to do the following:

- Make the initial runs as long as possible.
- At all stages, overlap input, processing and output as much as possible.
- Use as much working memory as possible. Applying more memory usually speeds processing.
- If possible, use additional disk drives for more overlapping of processing with I/O, and allow for more sequential file processing.