Title

A note on bit-mapped free sector management

Journal

ACM SIGOPS Operating Systems Review, 27(2)

ISSN

0163-5980

Author

Long, Darrell DE

Publication Date

1993-04-01

DOI

10.1145/155848.155851

A note on bit-mapped free sector management

Darrell D. E. Long
Computer & Information Sciences
University of California, Santa Cruz

The most common methods for maintain a list of free sectors on disk are to use either a *linked list* or a *bit map* [1].

Using a linked list has the advantage that is requires no extra storage since the links are stored in the free sectors. It also provides quick allocation and deallocation, requiring only that a free sector be removed from the head of the list, or a freed sector be added to the head of the list, respectively.

The main disadvantage of a linked list is that over time the list tends towards random. That is, unless the list is sorted, sectors are placed on the list in no particular order. The result is poor locality during file access, significantly impacting performance by increasing the average seek time.

By using a bit map, adjacent free sectors will always appear adjacent in the bit map. There is a small cost in terms of storage; that is, the bit map will contain one eighth as many bytes as there are sectors on the disk. There is a potentially more important concern: the average number of bits that must be scanned in order to find a free sector. Since this technique was first used, the size of disks has increased by approximately four orders of magnitude. If the number of bits to be scanned on average increased even a small fraction of this amount, the technique would need to be abandoned.

This question came up in our undergraduate operating systems course. Our initial speculation was that bit maps would be inappropriate for the large disks that are becoming available.

As the following analysis will show, the bit map technique remains viable for large disks, and as we shall see it is, in a sense, independent of the size of the disk.

Assume for the moment that the free sectors are uniformly distributed across the disk, resulting from files being freed in no particular order. Further assume that the system is in steady state with r sectors free on average out of a total n. The problem then reduces to drawing one of r cards from a deck of size n [2]. Let b_i, $1 \leq i \leq n$, be the bit map where there are r zero bits indicating free sectors. Then the probability of the first bit being zero is

$$p_1 = \Pr[b_1 = 0] = \frac{r}{n}.$$

Similarly,

$$p_2 = \Pr[b_2 = 0 \wedge b_1 = 1] = (1 - \frac{r}{n})\frac{r}{n-1},$$

and

$$p_3 = \Pr[b_3 = 0 \wedge b_1 = 1 \wedge b_2 = 1] = (1 - \frac{r}{n})(1 - \frac{r}{n-1})\frac{r}{n-2}.$$

In general,

$$p_k = \Pr[b_k = 0 \wedge b_j = 1, 1 \leq j < k] = \left[\prod_{i=0}^{k-2}(1 - \frac{r}{n-i})\right]\frac{r}{n-k+1} = \left[\prod_{i=0}^{k-2}\frac{n-i-r}{n-i}\right]\frac{r}{n-k+1},$$

which simply means that the probability of b_k being the first zero bit is conditional on the probability of $b_j, 1 \leq j \leq k-1$ being one. As k increases the likelihood of b_k being zero increases as the number of bits yet to be scanned decreases.

We can reduce this unwieldy expression by noticing that

$$\prod_{i=0}^{k-2}(n-i-r) = \overbrace{(n-r)(n-r-1)\cdots(n-r-k+2)}^{k-1} = \frac{(n-r)!}{(n-r-k+1)!}.$$

Similarly,

$$\prod_{i=0}^{k-2}(n-i) = \frac{n!}{(n-k+1)!}.$$

The result is that

$$p_k = \frac{(n-r)!}{(n-r-k+1)!} \times \frac{(n-k+1)!}{n!} \times \frac{r}{n-k+1} = \frac{(n-r)!\,(n-k)!\,r}{(n-r-k+1)!\,n!}.$$

This can be rewritten as

$$p_k = \frac{\binom{n-k}{r-1}}{\binom{n}{r}},$$

which can be viewed as the number of ways to obtain a run of length $k-1$ one bits divided by the total number of ways to arrange r zero bits out of n total.

Let the random variable x be the number of bits that must be scanned before finding a zero, then the expected value of x is

$$\mathbf{E}[\mathbf{x}] = \sum_{i=1}^{n-r+1} i p_i = \sum_{i=1}^{n-r+1} \frac{i\binom{n-i}{r-1}}{\binom{n}{r}}.$$

Since the only the first $n-r$ bits can be allocated, in the worst case the scan will have to continue to bit $n-r+1$ (which must certainly be zero). This sum can be reduced to

$$\mathbf{E}[\mathbf{x}] = \frac{n+1}{r+1}.$$

The variance of x is given by

$$\mathbf{V}[\mathbf{x}] = \frac{r\,(n-r)\,(1+n)}{(1+r)^2\,(2+r)}.$$

This means that the expected number of bits that must be scanned to find a free sector depends only on the ratio of the total number of sectors to the number of free sectors. Said another way, given a disk that is 90% full, it does not matter whether the disk is 20 megabytes or 5 gigabytes, the average number of bits that must be scanned is approximately 10 (except for a vanishingly small ϵ).

We assumed that free sectors were uniformly distributed across the disk. If the algorithm always scans sequentially from the first bit, this will almost never be true. It has been observed empirically that when this is done, allocated sectors will cluster at the start of the disk, and free sectors will cluster towards the end. The result is that the expected number of bits to be scanned will tend towards the worst case of $n - r + 1$.

There are two simple heuristics that alleviate this problem. The simplest way to maintain a roughly uniform distribution of free sectors is to begin scanning at a random position each time. The method more frequently used in practice is to use a *roving pointer*, where the next search begins where the last left off. It is also common to reset the roving pointer to the beginning of the deleted region when a file is freed. This increases the randomness of the free sectors since the roving pointer is reset to a random position just before a sequence of zero bits. It can also significantly improve performance as the disk becomes increasingly full, since the roving pointer will often be at the beginning of a free region.

While the analysis presented here is simple, we have been unable to locate it in any of the standard texts. It has important implications to practitioners, who might instead follow their intuition and implement a more complex scheme than necessary.

References

[1] A. S. Tanenbaum, *Modern Operating Systems*. Englewood Cliffs: Prentice Hall, 1992.

[2] B. V. Gnedenko, *The Theory of Probability and the Elements of Statistics*. New York: Chelsea, 1989.