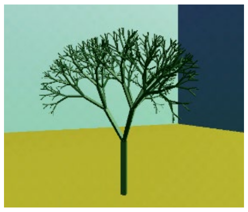


GATS Companion to du Project

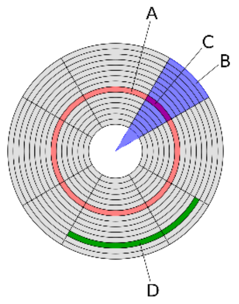
Author: Garth Santor
Editors: n/a
Copyright Dates: 2020-2021
Version: 2.0.0 (2021-03-29)

Overview

Background and FAQs about the du project.

Concepts

Disk/File allocation



A platter of your hard drive (HDD).

A = track (pink, ring)

B = geometric sector (blue, pie-wedge)

C = track sector (purple, intersection of pie, ring)

D = cluster (green, several tracks)

Efficiency – storage efficiency vs speed efficiency

- Every block must be recorded in the index.
- The smaller the blocks, the bigger the index, but less wasted space.
- The larger the blocks, the smaller the index, but more wasted space.

Block – definition

The smaller allocation unit on a drive. The minimum memory that must be consumed to store a file.

File size (bytes)	Block size (bytes)	#blocks	Waste
10,000	512	$\lceil 19.5 \rceil = 20$	272
10,000	2048	$\lceil 4.8 \rceil = 5$	1808

How to get a cluster (block) size in Windows

```
#include <Windows.h>

// Determine cluster size
auto drive = std::filesystem::absolute(rootFolder).root_name().string() + '\\';
uintmax_t clusterSize;
DWORD sectorsPerCluster, bytesPerSector, numberFreeClusters, totalNumberOfClusters;
if (GetDiskFreeSpaceA(drive.c_str(), &sectorsPerCluster, &bytesPerSector, &numberFreeClusters, &totalNumberOfClusters)) {
	clusterSize = uintmax_t(sectorsPerCluster) * bytesPerSector;
}
```

How to get started?

Consider:

- Can I use <filesystem> to navigate up and down a folder tree? (i.e. recursion)
- Can I use read command-line arguments?
- What information do I need to collect?
- How do I store the information collected? Or in what do I store the information collected?

Support Files

The archives:

- dutest folder (files, nosubfolders).zip
- dutest folder (files, subfolders).zip

Contain the folders and files that dutest generates. You can use them to test your program.

Use 'dutest -v' to see the full output of the test program comparison.

FAQ

Q: Sometimes the size is exactly like File Explorer, but sometimes it is different. Why?

There are two reasons for the differences:

- Compression
- Small file optimization with the MFT

If you have a compressed folder – the size-on-disk could be significantly smaller than the file size.

One of the problems that has plagued hard disk optimization, is that optimizing for large files is bad for small files, and optimizing for small files is poor for large files. NTFS (the Windows NT through Windows 10 file system) is better optimized for larger files, but has a trick for very small files; store them in the MFT (Master File Table).

What is the *master file table*?

The Master File Table is a type of file contained in the NTFS filesystem. It isn't a user visible file, it's a file that stores information about the other files. It contains information such as the size, time and date stamps, permissions, and other metadata (music files could store length, sample rate, etc.) A lot is stored in each record. However, most is necessary for large multi-cluster files. When a file is very small (a fraction of a cluster), the unused elements of the file record can be used to store the file itself. This is usually the case for files under 512 bytes. As a result they don't consume extra clusters of the disk.

What should our program do?

Treat all files the same way, use 4KiB (4096 byte) clusters, or use the code provided in this document to get the volume cluster size. Therefore, we will treat small files the same way we treat all files. Allocate at least one cluster/block.

Q: Can we use third-party libraries?

No, only C++ 17 standard libraries, or <Windows.h>

Q: There seems to be rounding errors when calculating the block size of large files?

This could be the case when using *ceil()* and real number division for huge files (remember, doubles only have about 15 digits of accuracy).

Solution: perform the block size computation using only integer arithmetic. This means that you will have to use integer division and modulus to compute quotients and remainders.

The block size is the quotient of dividing the file size, by the cluster size. If there is a remainder, you'll need to bump up the block size by one.