Technical Report

UCAM-CL-TR-3
ISSN 1476-2986

Number 3

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

A replacement for the OS/360 disc space management routines

A.J.M. Stoneley

April 1975

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

TECHNICAL REPORT No. 3

A REPLACEMENT FOR THE OS/360 DISC SPACE MANAGEMENT ROUTINES

by

A.J.M. STONELEY

University of Cambridge Computing Service

Series Editor:

M.F. Challis
University of Cambridge
Computer Laboratory
Corn Exchange Street
Cambridge CB2 3QG
England

April 1975

Summary

In the interests of efficiency, the IBM disc space management routines (Dadsm) have been completely replaced in the Cambridge 370/165.

A large reduction in the disc traffic has been achieved by keeping the lists of free tracks in a more compact form and by keeping lists of free VTOC blocks. The real time taken in a typical transaction has been reduced by a factor of twenty.

By writing the code in a more appropriate form than the original, the size has been decreased by a factor of five, thus making it more reasonable to keep it permanently resident. The cpu requirement has decreased from 5% to 0.5% of the total time during normal service.

The new system is very much safer than the old in the face of total system crashes. The old system gave little attention to the consequences of being stopped in mid-flight, and it was common to discover an area of disc allocated to two files. This no longer happens.

A.J.M. Stoneley

April 1975

Contents

A Replacement for the OS/360 Disc Space Management Routines.

A.J.M. Stoneley.

1 Introduction

In a large general purpose operating system, particularly one which supports a substantial number of active terminals, the creation and deletion of a disc file, whether a permanent file or temporary work space, cannot be regarded as a rare event. The Cambridge Phoenix system, evolved from TSO, currently supports over forty simultaneously interactive terminals in addition to the normal batch or offline load. It became apparent at an early stage that the OS disc space management routines were a significant overhead and a severe bottleneck to such activities. The main embarrassment stemmed from the number of inevitable disc transfers, typically sixteen to twenty per transaction. Apart from the load on the device, this demanded a long real time in core, and even longer waits for processes held up for the necessary serialisation. A lesser but present embarrassment was the unsafe nature of the routines. System crashes at unfortunate moments could and did result in areas of disc being attached both to the free area and to the allocated area. The size of the code was such that it was impractical to keep all of it permanently in core. The consequent paging amplified the disc traffic. With these points in mind, the disc management routines were completely redesigned.

2 The Original System

Under OS/360, all data relating to the allocation of space on a particular disc volume, or pack, are held in a reserved area of that pack known as the VTOC, or Volume Table Of Contents. The VTOC is formatted with a large number of fixed length keyed blocks, each 140 bytes long. The hardware can read or write particular blocks and can also scan the keys for a particular one, although such a key search is usually a lengthy operation. Every file on the volume, whether permanent or temporary, is represented by one or two of these blocks. In the key of the first block, the Format 1 block (F1), is the name of the file, so that this block can be found by key search, although normally the actual address of this block is known from other sources, such as the catalogue. The body of the F1 contains file characteristics, such as blocksize, and up to three 'extent descriptors'. An extent descriptor demarcates a continuous area of disc of arbitrary size, and so the F1 can describe a file consisting of up to three disjoint areas of disc. Up to thirteen more of these descriptors may be accommodated in the second block, the Format 3 block (F3), which is pointed to by the F1. The limit of sixteen extents per file is built very solidly into various other parts of OS and the obvious extension of this chain of descriptor blocks is not made.

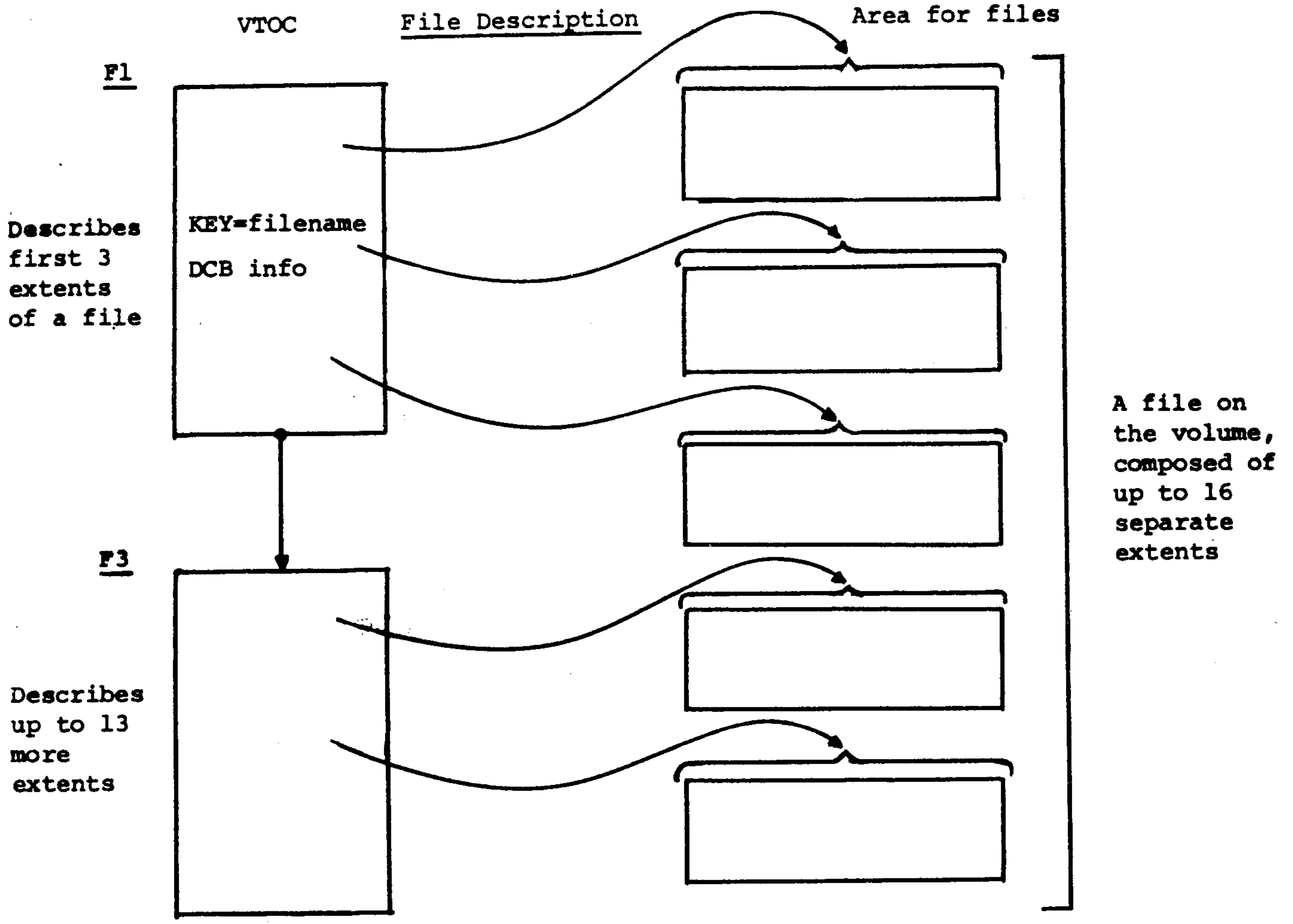
The free space is described by a separate chain of VTOC blocks, the Format 5 blocks (F5). The first F5 of the chain is located at a standard position in the VTOC. The F5s are filled with extent descriptors, in a different and more compact format, one F5 containing 26 such descriptors. The VTOC itself is described by a Format 4 block, or F4, which is very similar to an F1.

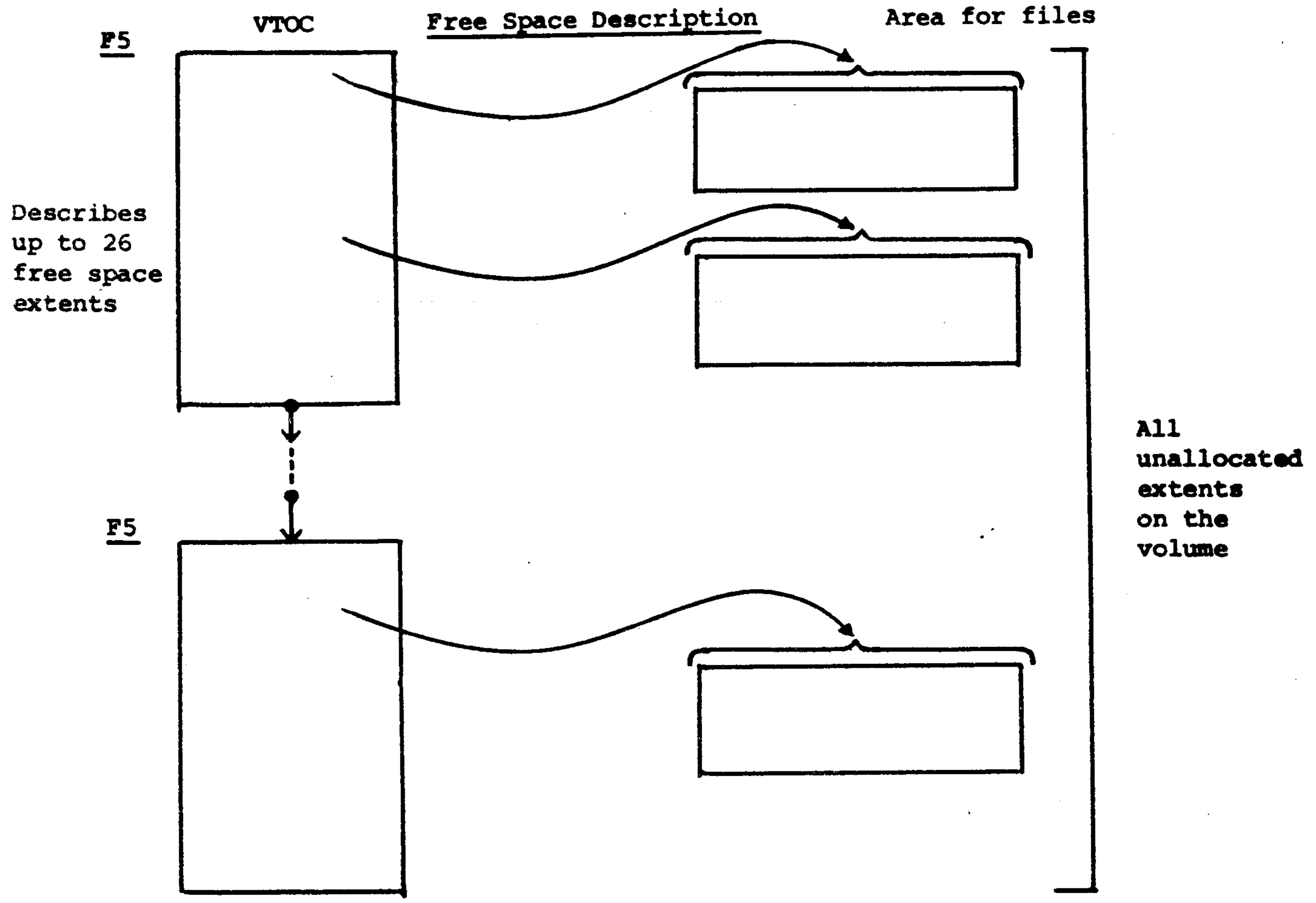
Free blocks in the VTOC are distinguished by having zero key.

The space management routines provide for the creation, extension, deletion and contraction of files, the programs for each of these functions being quite distinct, despite the number of common operations. In each case the F4, all the F5s, the F1 and the F3, if any, are read down and written back, except that during creation a key search of the entire VTOC is made to check against duplicate names. Whenever a free block is required, it is obtained by key search for zero key.

Long before the major redesign, the duplicate name search was eliminated, since it was rendered unnecessary by various other local changes. Also, whereas the F1 was retrieved by key search during deletion in the original code, provision was made to pass the address of the F1 to the deletion routine whenever possble, in fact in most cases. Key searches were still used to retrieve free blocks. The disc traffic required was still very high. Typically the VTOC contained about eight F5s, and so some 20 disc transfers and a key search were quite usual.

There are other complications which are not relevant to the Cambridge system, since they relate to unused facilities.


VTOC
File Description
Area for files
F1
Describes first 3 extents of a file
KEY=filename
DCB info
F3
Describes up to 13 more extents
A file on the volume, composed of up to 16 separate extents


F5
VTOC
Free Space Description
Area for files
Describes up to 26 free space extents
F5
All unallocated extents on the volume

3 The replacement

When modifying a part of an operating system, it is necessary to maintain the interfaces with the rest of the system. In this case the interfaces are of two kinds. The obvious interface is the call of the space management routine, in which an array of arguments is passed. The less obvious one is the VTOC itself, inasmuch as the routines for opening and closing files also interact with the VTOC. In particular, the physical block structure of the VTOC and the contents of the F4, F1s and F3s must remain the same. The parts which are completely the preserve of the space management routines are the F5s and the free blocks. This is not so great a hardship. Reading and writing F1s and F3s is a small part of the disc traffic, and in the common case there is no F3. The embarrassments stem from those parts of the VTOC which are local to space management, the F5s and free blocks.

In the redesigned system the free space on a volume is represented in an array of bits, a bit map, in which there is one bit for each unit of space, this bit being one if the unit is allocated and zero otherwise. The blocks of the VTOC are similarly represented. Normally these maps are permanently in core, but they can, if necessary, be paged as a single block per volume onto an ordinary file, possibly on the high speed drum. Assuming that this is not necessary, the search for free space and free VTOC blocks now involves no disc traffic at all, and the only disc transfers during space management are the reading and writing of F1s and, less commonly, F3s.

The maps are not preserved across system restart. Whenever the system is re-initialised, the VTOC is scanned completely and the maps are reconstructed by reference to the existing F1s and F3s. This scan does not take as long as might be imagined. It is actually possible to read a complete trackfull in one revolution of the disc, and this is done. The VTOC is thus scanned in about a second of real time.

The code is written as an integrated block, so that the four main functions, create, delete, extend and contract, can share many common subroutines. The resulting code, about a fifth of the size of the equivalent original code, is sufficiently small that it can be kept permanently in core. The space taken up by the maps is substantially offset by the work space used by the original routines.

It should be noted that some of the more baroque features of OS have not been implemented. Amongst these are split cylinder allocation, suballocation, and indexed sequential organisation. These features are, however, available on 'private' volumes, which are managed by the original routines.

4 Details of operation

There is a permanently running key 0 job, called DM, with a region of its own. This region contains the maps, workspace, and most of the code associated with the DM. Much of this code, however, is executed by the requesting task, rather than the DM task. The functions of the DM task are to initialise the system and to do the paging of the maps, together with organising page frames.

The DM is started by a start command, issued automatically after IPL. The first load is the main body of code and remains resident. The load module name is IEFDSO, and in consequence the job has key 0. (This privilege is a standard OS one, conferred on started tasks with particular entry point names.) The first action of the DM is to LINK to the initialisation module. On return from initialisation, DM GETMAINs enough space for its page frames, the initialisation module having been flushed. It then awaits and services paging requests.

The initialisation module reads a list of volume ids from the PARM field, finds the volumes and associates a Vbase with each. A sufficient supply of Vbases is assembled into the main module.

All Dadsm activity is now locked out by ENQing on initialisation, and the paging file is opened and formatted.

For each volume specified:-

> A proforma Vpage for this device type is found. These proformas have been link edited into the initialisation module. The first part of the proforma is completed, so that the common DM subroutines can now operate, using this proforma as a Vpage. In particular the VTOC IO routines can work.
>
> The VTOC is now scanned and the bit maps are initialised accordingly. Any F5 blocks are erased so that the volume will appear full if it is at some time accidentally accessed by the OS space management routines. The high water mark is set to the top of the VTOC (key searches are very rare). The VTOC is read by tracks, with two track buffers and overlapped IO.

When all volumes have been dealt with, the address of the DMbase is planted in CVT3DISC, an element in the tertiary CVT pointed to by CVTUSER, and the initialisation module is left.

The interfaces to the outside world are, of course, standard SVCLIB modules. The function of these modules is to decide whether to use DM or Dadsm. In the former case, they provide a buffer between the various OS conventions and the uniform DM conventions. In the latter case they call the normal Dadsm modules. The gross structure of the interface modules is:-

```
$( ENQ against DM initialisation

   if DM is present and knows this volume
     then $( ENQ for this volume with SMC=STEP
             BALR to DM entry gate
             DEQ off volume with RMC=STEP $)
     else link to Dadsm  fi

   DEQ from DM initialisation  $)
```

Interface modules find the DM by way of CVT3DISC, which points to the DM base if DM is ready and is zero otherwise. DMbase contains a pointer to a chain of Vbases, one for every volume under DM control. Apart from identifying a volume, the Vbase contains any data specific to that volume which may be required before passage through the DM entry gate. Once through this gate, Vbase points to a page frame, the Vpage, containing the bit maps and other pageable data and tailing off into non-paged work space.

The function of the entry gate is to set up the Vpage and the standard registers and to exit to the required function: allocate, scratch, extend or release. It also preserves the link and R1 in the Vbase.

The converse functions are performed at the exit gate.

Once within the main body of code, an orderly array of routines is available. The fundamental ones are those for testing and setting bits within the bit maps, searching for free areas within the maps, and reading and writing blocks of the VTOC. Internally, all work is in terms of track numbers and VTOC block numbers, curiousities such as CCHHRs and TTRs being confined to the external interfaces. A set of routines for converting between all these forms is provided. There is then a higher level set of routines, such as one to find and allocate an extent of given length. With the help of these routines, the mainline code for each of the four major functions, allocate, scratch, extend and release is then in each case only a few pages long. For example, allocate, by far the most complicated, occupies only four pages of text. This is important in rendering the program comprehensible.

Because errors in managing disc space can cause large scale loss of data from disc, the code is liberally endowed with self consistency checks. As an example, the routine for setting a bit in a map will refuse to do so if that bit is already set. All errors are signalled to a central error routine, which informs the operators of the error, and suggests remedial action, such as running the VTOC mending program. If the error is sufficiently serious that the contents of the disc are thought to be at risk, the system is forced to an immediate halt.

5 Conclusion

It has proved possible to replace the OS/360 disc space management routines by a system which is far more appropriate to the way in which OS is used at Cambridge. In particular, it is now relatively cheap to create and delete large numbers of small files. Certain features, not used at Cambridge and inappropriate to a large time shared system, are not provided, although they can be made available for private discs.

The efficiency gains are illustrated in the following table which shows the real and cpu times required to allocate and then scratch a single one track file using the original and new systems. The disc pack used here was one of those normally available for permanent and temporary files during the running of the Computing Service. For this experiment, all relevant code was permanently resident in core and the system was otherwise idle. During normal running of the service, Allocate and Scratch are each entered about once every two seconds.

seconds	OS code	replacement
cpu	0.11	0.012
real	2.9	0.15

(The OS code used here itself includes some improvements. The duplicate name search is eliminated from allocation and the disc address of the F1 is passed as a parameter to scratch so as to eliminate a key search.)

Amongst the more obvious advantages, the reduction in the real time required has almost eliminated a severe system bottleneck (the Q4 bottleneck), caused by the serialisation of allocation under OS.

The new system is safe in the face of unexpected halts of the computer. Using the original code, it was common to discover after a system crash that a VTOC was in an unsafe non-standard state. This has not happened since the introduction of the new system.