Inter-Office Memorandum

To Pilot W.G.

Date September 30, 1977

From Peter Bishop

Location Palo Alto

Subject Unique FileIDs
Are 64 bits enough?

Organization SDD

XEROX

Filed on: <Bishop>uid.memo

There has been some controversy over whether fileIDs provided by Pilot should be unique over all Xerox systems. The original Pilot Concepts and Facilities [1] specified that a FileID would contain 64 bits and would be "unique over all time". The first draft of the Pilot Functional Specification [2] merely said that a FileID would be "unique over the life of the system". During the file working group meetings that followed the first draft, however, it became apparent that it might be valuable to have the FileID unique over all time and space. We only have the power to make this cover all Xerox OIS systems, however. This seemed particularly appealing since it did not appear necessary to increase the size of the FileID significantly in order to achieve this added generality. Accordingly plans began to be made to use a 64 bit FileID that would be unique over all Xerox OIS systems that use Pilot [3]. Some people have had reservations about the need to go to the effort that is needed to ensure that the FileIDs be unique over all Xerox systems [4]. Others have felt strongly enough about the issue, however, to produce a document that sketches some of the issues involved [5]. The issue is currently unresolved as can be seen from the current draft of the Pilot Functional Specification [6] which does not specify what the range of uniqueness is, but assumes that if a file moves to another system element it may be possible for software to automatically discover its new location. This note proposes mechanisms for making FileIDs unique over all Xerox systems without using more than 64 bits for the FileID.

Why Span All Xerox Systems?

The justification behind the use of FileIDs that are unique over all Xerox systems is that it satisfies a need that would otherwise exist to translate between the separate ID spaces on different Xerox systems. When two computer systems are tightly coupled, it is obvious that they should share a single fileID space (this might be implemented merely by prefixing a system identifier to the fileIDs normally used on each system). By providing a single ID space for all systems, however, we provide three features: 1) systems can be dynamically reconfigured easily, 2) disk packs can be moved from system to system easily, and 3) the size of the fileID is independent of the number of systems that are tightly coupled together. The cost of providing fileIDs that are unique over all Xerox systems is in two parts: a) the FileID must be somewhat larger than if each system had its own fileID space (this cost is surprisingly small, however), and b) allocation of fileIDs must be done more carefully to prevent one system from allocating fileIDs that are being allocated by another system that may reside in a foriegn country. The reason the storage cost for a general fileID is so small is because the fileID for a single system must be rather large and the number of files that can be handled by a single ID space increases exponentially with the size of the fileID. Since the total amount of storage used by fileIDs is reasonably small, the cost of the larger fileID is not severe even if the fileID must be as much as a factor of two larger. In fact, the increase in the size of the fileID in order to span all Xerox systems is probably less than 30%.

General Approach

Whether the fileID spans only a single system or all Xerox systems, it is necessary to choose a size for the fileID that is large enough to serve the range of systems that Xerox would like to provide. Actually the range of the performance of the systems Xerox would like to provide is rather broad and the rate of generation of fileIDs is very hard to predict. Rather than trying to derive the appropriate size of a fileID, I will instead propose a size that I consider to be adequate and then defend the adequacy of this size. The basic technique I will use is to determine an upper bound on the rate of allocation of fileIDs and provide enough fileIDs to last for the lifetime of the product. Thus I do not pretend to have found a least upper bound on the number of fileIDs that are needed, I merely suggest that there is a high probability that the size proposed is larger than the true least upper bound on the number of fileIDs that will be allocated during the lifetime of the OIS product line.

How Large is 2^{64}?

The size of unique fileID that I would like to defend is 64 bits. The basic reason that I feel that this size is adequate is because when 2^{64} is divided by 10^6, the resulting number is the number of milliseconds in 500 years. This means that using a 64 bit fileID, we could support one million simultaneously and continuously operating processors for 500 years, assuming that the long-term average rate of file creation on a single processor is one per millisecond, or 3.6 million files per hour. One million systems would provide about 2% of the entire work-force in the United States with a personal computer.

The Size of the Excessive Safety Factor

The above numbers are all very conservative worst-case estimates. In fact, they are so absurdly conservative that we can estimate an excessive safety factor that could be removed from the above estimates while still providing an adequate safety factor in the use of fileIDs. The worst-case estimate of total product lifetime of 500 years is probably too large by a factor of 10 (we should expect to be using a different operating system in at least 50 years, at which point the number of running systems will decrease as we switch to the new system). Similarly, the worst-case long-term average of file creation is probably too high by a factor of 100 even assuming that systems do actually operate continuously. Finally, the assumption of continuous operation is ridiculous when considering personal computers. Even time-shared systems do not actually run more than about 1/4 of the time, thus this estimate is too large by a factor of 10. Thus we find that it is almost certain that 2^{64} is more than 10^4 times the total number of files that will be created on all OIS products (that run Pilot) combined.

The Problem of Allocating FileIDs

Some of this excessive safety factor must be used to ensure that no two OIS systems allocate the same fileID. There is a fundamental requirement that there be a single, central place in the world that is the highest authority for allocating fileIDs. This is obviously unacceptable for the allocation of individual fileIDs, however. It is possible, however, for the central allocator to delegate allocation authority over portions of the ID space to less central allocators. Each allocator thus should have a range of fileIDs that it may allocate. One allocator may allocate a subrange of fileIDs to another allocator, but only if the original allocator then behaves as if all of those fileIDs have been allocated to actual files.

The central allocation problem is solved by finding a system of delegating allocation authority to progressively less centralized allocators. A system of allocators consisting of three levels seems to be adequate. The Xerox factory is, of course, the highest level. Actually, there may be several Xerox factories, thus requiring that some care be given to the allocation problem at this level as well. The second level of allocator should be associated with the individual processor. It can be argued that there is no need for any allocators below the level

of the individual processor, but it would be nice for a fileID to give a hint about what disk volume it resides on. Such a hint can be obtained if each disk volume contains the lowest level allocator of fileIDs. Thus the allocator in each system allocates a series of fileIDs to a disk volume.

Crash-Proof Allocators

A basic problem with a multi-level fileID allocator arises if it is possible for an allocator to be destroyed. When an allocator is destroyed, we must assume that all of the fileIDs it was responsible for were allocated. Thus the higher the level of the allocator, the more serious is the destruction of the allocator because the destruction of the allocator effectively allocates more fileIDs. In addition, it is particularly painful if the allocator that resides in a processor is destroyed because the operation of allocating fileIDs from a Xerox factory to the processor is a very slow, expensive operation. Thus it is necessary for the allocator in the processor to be able to survive system crashes of all kinds. This requires the allocator to be stored in non-volatile memory. Current technology would allow a small portion of the system hardware to be devoted to providing a special counter in EAROM (electrically alterable read-only memory). Current technology would only allow such a counter to be incremented a total of 10^6 times, but this limitation can be programmed around. If we have designed the allocation system so that high level allocators are not lost, then the allocator that is at the factory should be designed very carefully as well.

The System of Allocators

Since we would like to handle a maximum of 10^6 simultaneously operating processes, it seems prudent to allocate only 2^{40} fileIDs to each processor. This allows us to allocate fileIDs from the factory a total of 2^{24} (or 16×10^6) times before we have exhausted our ID space. We can only allocate as many as 2^{40} fileIDs to a processor if we are reasonably sure that the allocator in a single processor will not be destroyed.

The next question is how many fileIDs should be allocated to a disk volume at one time. If we are too generous, there is an opportunity here for fileIDs to be allocated to allocators that will never allocate the fileIDs to actual files. We must ensure that at least 1% of fileIDs allocated to leaf-node allocators (i.e. disk volumes) actually be assigned to files. Since a processor is tightly coupled to its disk, it would be possible for the processor to allocate arbitrarily small blocks of fileIDs to disk volumes. If the processor allocates too few fileIDs to a disk volume at once, however, then the fileID will not provide any useful hint about the location of the file. These two factors seem to be well balanced if we allocate fileIDs to disk volumes in blocks of 2^{16} fileIDs. 1% of these fileIDs will be assigned to actual files if 600 files are created on the volume or if, on the average, 1% of disk volumes allocate all 2^{16} fileIDs.

This allows a processor to allocate blocks of 2^{16} fileIDs to disk volumes a total of 2^{24} times before the total set of 2^{40} fileIDs has been exhausted. A processor could initialize disk volumes at the rate of one per second continuously for 6 months before exhausting its allocation. If the long term average rate of initializing disk volumes is as low as one per minute, however, then the processor's allocation of 2^{40} fileIDs would last for 30 years.

What happens if, for some strange reason, a processor does in fact use up its entire allocation of fileIDs? It will probably be necessary to replace the EAROM chip at this point because its counter will have been modified as many times as possible. At the same time as the EAROM chip is being replaced, a new series of 2^{40} fileIDs must be allocated to the processor. Both of these operations must be performed by Xerox maintenance personnel. It is the desire to reduce the need for this maintenance operation, especially on high performance systems, that causes us to allocate as many as 2^{40} fileIDs to each processor. It

seems prudent to replace the EAROM chip when 80% of the fileIDs have been allocated. The resulting percentage wastage of fileIDs is very small.

Representations for Allocators

Although this analysis seems reasonable, it is quite clear that there is no very good basis for choosing 2^{40} fileIDs as the number that should be allocated to a processor. The choice of 2^{16} fileIDs per disk is also a pure guess. These guesses are good enough to start from, but we should maintain the ability to change the number of fileIDs that are allocated at once to a processor and to a disk volume. It may even become necessary to allocate different numbers of fileIDs to different processors or different disk volumes. If we keep track of the date on which an allocation of fileIDs was made to a processor or a disk volume, then the number of fileIDs that should be allocated when this initial allocation is exhausted can be automatically calculated from the observed average rate of usage of fileIDs combined with the length of time desired before another allocation of fileIDs to this processor or disk volume is needed.

The allocator that resides on a disk volume should thus contain four numbers: a) the first fileID in the sequence allocated to the disk, b) the maximum fileID in the sequence allocated to the disk, c) the next free fileID, and d) the date at which the sequence of fileIDs was allocated to the disk volume. Once this information is maintained, we have a great deal of flexibility to construct allocators that will waste very few fileIDs. Newer, better allocators could be put into a system merely by changing a few small programs, i.e. the allocators themselves.

The allocator contained in the processor has a different set of design constraints and so should use a slightly different representation for the allocator. First, since the EAROM can only be incremented a total of 10^6 times, a group of fileIDs must be allocated at once. The number of fileIDs to be allocated at once should be contained in the EAROM. Second, bits in the EAROM will probably be at a premium. The allocator in EAROM must contain at least three numbers: a) the next free fileID, b) the last free fileID, and c) the number of fileIDs to allocate at once. We can save storage for the low order 16 bits of all three numbers by forcing the minimum allocation of fileIDs from the processor to be 2^{16} and then aligning the allocations on 2^{16} boundaries. If the maximum number of fileIDs that a processor allocates at once is 2^{24}, then the number of fileIDs allocated at once can be represented with 8 bits. If the maximum number of fileIDs allocated to a processor from the factory is set at 2^{40} and the minimum at 2^{24}, and if the next free fileID is represented by 48 bits (a 64 bit number on mod 2^{16} boundary), then the last free fileID can be represented with 16 bits (a 64 bit number on mod 2^{24} boundary with the high order 24 bits derived from the next free fileID). This allocator is exhausted if the next allocation would result in the next free fileID having bits 24-40 equal to the 16 bits used to represent the last free fileID. The other two numbers that a disk allocator maintains (the first fileID in the original sequence and the date the sequence was allocated to the disk) need not be kept in the EAROM. The maintenance personnel should make an attempt to keep these numbers, but the system should also try to keep these numbers on the system disk. If a crash wipes out these numbers on disk, it would be acceptable to replace them with the next free fileID and the date of the crash. These two numbers still allow a rate of fileID usage to be calculated even if they only give the rate of usage since the last major crash.

System software has the option of allocating fileIDs to disks in units smaller than or larger than the number allocated at once by the hardware allocator, but a system crash may destroy the software allocator. If the allocator maintained by system software is destroyed, it must be re-initialized from the hardware allocator in the processor. This initialization should be done only when the system software actually needs to allocate fileIDs rather than during an initialization phase of crash recovery.

Conclusions

The software that attempts to estimate on which disk volume a file resides should be written so it will work when different volumes are allocated different numbers of fileIDs. The limitations of this software will probably determine whether the fileID allocation strategy can be changed easily or not. Since we do not know at this time how sophisticated the allocation of fileIDs needs to be, this software should be written with enough generality to allow very sophisticated allocators. By the time we have allocated fileIDs in blocks of 2^{40} to 100,000 processors, we will have used up less than 1% of our fileID space but we will have gained enough experience to be able to redesign the system of allocators (if necessary) so that fewer fileIDs are wasted (allocated to allocators which then do not allocate them to actual files).

Although I mentioned earlier that the percentage of storage used for fileIDs is not very large, the current plans include storing a fileID in the header of every disk page. Thus the total amount of storage used for fileIDs will be considerable even if it is only 3% of the total storage. We should make an attempt, then, to keep the size of the fileID small without placing the future utility of the fileIDs at risk. I hope that I have shown that the fileID does not need to be larger than 64 bits. If the scheme seems to require mildly complex hardware, the savings in storage should cover any added hardware cost. I would like to point out that similar problems occur even if the fileID is unique over a single system. Most of the fileID is designed for the use of a single system. The existence of the high order bits in the fileID allows the number of fileIDs that are normally allocated to a single system to be smaller than if we were guaranteeing uniqueness over a single system. We need not be concerned about the possibility of the 105^{th} processor that is delivered being kept by its owner for 200 years. Thus removing the requirement of uniqueness of fileIDs over all Xerox systems will not significantly reduce the costs of managing the allocation of fileIDs or even significantly reduce the costs of storing fileIDs.

References

[1] *PILOT: The OIS Control Program - Concepts and Facilities,* XEROX Information Technology Group, SDD, August, 1976.

[2] *Pilot: The OIS Control Program - Functional Specification,* Version 1.0, XEROX Information Technology Group, SDD, December, 1976.

[3] McJones, Paul, "B-file Issues", Memo to the File Management Working Group, SDD/SD, April 28, 1977.

[4] Lauer, H. C., "Unique names for file identifiers", Memo, SDD/SD, July 15, 1977.

[5] Gifford, David K., "Unique File Identifiers", Memo, SDD/SD, July 20, 1977.

[6] *Pilot: The OIS Control Program - Functional Specification,* Version 2.0, XEROX Systems Development Division, September, 1977.