WHY USE CACHE:

"PERFORMANCE IMPROVEMENTS"

OS/3 DISK CACHE FACILITY

Jeff Snyder
Sperry, Blue Bell

I will be discussing the OS/3 Disk Cache Facility as it is supported on System 80 Models 4, 6, and 8. I am directing my discussion of cache towards the general operation, initialization and control of the program and I will highlight differences between the Model 4/6 and Model 8 cache where they exist. If you have any questions during my discussion please bring them to my attention.

The "cache" process was designed to enhance the performance of OS/3 by alleviating the bottleneck of slow access to disk data. This bottleneck is caused by the physical limitations and constraints of the system hardware and software namely:

- o System overhead in handling I/O requests
- o Queueing delay as I/O requests must wait issuing
- o Disk seek time (moving accessor arm)
- o Latency (waiting for rotation)
- o Data transfer time

The cache removes this bottleneck by buffering recently used disk data in main memory. This approach is effective because,
a) information which has been used recently is likely to be reused and, b) information 'near' the information currently in use is likely to be used in the near future. Consequently, operating system overhead and queueing delays are greatly reduced while disk seek time, latency and data transfer time from disk to main memory are eliminated when the data being requested resides in a cache buffer.

This has a direct relevance within OS/3 because our standard libararian format is sequentially oriented with blocks at 256 bytes so that module or file processing consists of reading many blocks one at a time. This structure affects many OS/3 systems software programs such as the Librarian, Linkage Editor, Language Processors, Output Writer and Editor.

User programs which process sequential files would also be directly affected by the OS/3 cache. The other benefit of a cache which buffers most recently used disk data is that certain portions of a file (i.e. the index partition) might reside in cache for long periods of time, thereby increasing performance.

Generally, this caching mechanism works by taking a one or more block read and actually doing a multi-block read into a predefined cache buffer (if the block did not already reside in cache) and then passing the record back to the caller. Thereafter, all reads of blocks near that one will be moved from the buffer to the caller without disk access. I will explain this mechanism in more detail later in my discussion.

The OS/3 Cache consists of a symbiont and load module delivered as programs within the OS/3 Released system software programs. The Cache symbiont SL00CM is loaded by systems load module by symbiont initialization during IPL or by symbiont initialization via CM command. The cache configuration is controlled by an entry in the $Y$SDF system file with the following format:

SLIDE 1

The symbiont reads this record and uses the name as a guideline to define the buffer size and load the load module. The last four characters of the name are examined for values of 0000 or 0100 (or 16 for Mod 4/6) through 1024. All other values are invalid and cache would not be initialized. The values of between 100 and 1024 define the size of the cache buffer when multiplied by 1024 bytes. The value 0000 signals the symbiont to request the buffer size via console message. The message would be:

SLIDE 2

The operator replies NONE if cache is not to be used or a number between 100 and 1024 to be used as mentioned earlier.

Once the cache buffer size is determined, the memory is allocated. The buffer resides in the high end of memory if cache was initialized at IPL time, otherwise allocation is on a first fit basis. An error message is posted if the buffer cannot be allocated. The buffer and tables are then initialized.

SLIDE 1

	MODEL 8	MODEL 4/6
DEVICE ADDRESS	IMPL	IMPL
TYPE/FEATURE	CAC8	CACH
MICROCODE NAME	CA800000	CAC50000 CA740000

SLIDE 2

CM01 ENTER THE NUMBER OF 1024 BYTE BLOCKS OF MEMORY FOR DISK CACHE OR NONE. VALID VALUES IN THE RANGE OF 100 - 1024.

The entry in $Y$SDF is initially delivered with the operating system to force the outputting of the cache block # message (i.e., 0000 in the last four characters). This entry can be easily updated to include a block number by using SDU (System Definition Utility). This menu driver utility allows for manipulating entrys in $Y$SDF so you can change the Cache name according to your processing needs. The last four characters would be changed to values between 0100 and 1024.

The number of blocks message will be output every time cache is initialized until you updated the cache entry in $Y$SDF. I would recommend not changing the SDF entry until you have come up with the optimum size for your configuration.

The cache is now processing as an integral part of OS/3. You need not concern yourself with cache now that it is processing but if you do there are commands for manipulating that processing.

SLIDE 3

The operator may remove DCF (Cache) from the system at any time through use of the CANCEL command. The cache symbiont, SL@@CM, is cancelled by keying-in the following command at the console:

```
      CA CM,S,N
```

A message is posted:

```
      CM06 Disk Cache operation terminated.
```

The memory allocated to the symbiont and the buffer will be returned to the system for other utilization.

The operator may manually initialize Cache at any time, if is not already in the system, by keying-in the CM command at the console. The cache symbiont, SL@@CM, is then loaded and proceeds to initialize the cache via the CM01 message or via the default $Y$SDF entry. However, unlike supervisor initialization, the cache buffer is not allocated at the high end of memory. The MI MM console command will display where the buffer is located.

SLIDE 3

```
o  CA    CM,S,N

o  CM

o  ØØ  CM REMOVE DVC# [,DVC#,....DVC#]

o  ØØ  CM ACTIVATE DVC# [,DVC#,....DVC#]

o  ØØ  CM STAT
```

The next two commands are supported on the Model 8 only. Disks are always initialized to be cache candidates. However, the operator is provided with the capability to control cache operation to be selective by device. To inhibit cache activity for a particular device and remove all cache entries for that device, the following operator key-in is used.

```
     00 CM REMOVE dvc# (,dvc#, ...,dvc#)
```

The REMOVE command must be used for disks that are updated by more than one Model 8 System so that consistent copies of data are maintained to preserve integrity.

The operator may reactivate a drive that has been removed from the cache by keying-in the following:

```
     00 CM ACTIVATE dvc#  ,dvc#, ...,dvc#
```

The operator may monitor cache operation by keying-in an unsolicited input message to the cache symbiont in the following format:

```
     00 CM STAT
```

The following cache statistics will be printed at the console. They reflect the environment since the last code initialization.

```
     CM21 total number disc hits:
     CM22 number read hits:
     CM23 number writes:
     CM24 number search equals:
     CM25 number search HI/EQ:
     CM26 number READ ERROR
     CM27 number READ not cached:
     CM28 Search Hit
     CM29 number write through:
     CM30 number unreferenced IOS:
```

Internal Processing

The internal processing of code is directly related to the size of a block within the cache buffer. The block size is defined by the buffer size as dictated when cache was initialized. A diagram shows the correspondence.

CACHE BLOCKSIZE

CACHE BUFFER SIZE	CACHE BLOCKSIZE	
	MODEL 8	MODEL 4/6
512 TO 1024	48	60
256 TO 511	24	30
100 TO 255	12	10

This equates to saying, for example, that if 1024 was selected as the cache size then every time one 256 byte record was requested, and was not in the buffer, a maximum of 48 would be read. This is qualified by the fact that only one track of data is ever handled in one block.

Also, take special note of the situation at the 256 and 512 buffer size specifications. An example of differences in 511 and 512 show my point, at 511 you have almost twice as many blocks of half the size as you do at 512. This situation could show drastic differences in performance by only changing the used memory by 1024 bytes. Note the 511 could be better for interactive processing while 512 is better for batch.

The implementation of this cache in OS/3 was made comparatively easy because all disk I/O's must come through a common supervisor call, EXCP instruction which passes control to the Physical I/O Control Services for action. The commands are then diagnosed to see if the I/O is a cache candidate. The only candidates are:

- o Read Data
- o Write Data
- o Search, Read, Greater or Equal (Mod 4/6 R8.2 only)

When one of these commands is found the cache symbiont is given control. If the command is not one of these, the I/O is initiated as it would if cache was not present. Once the symbiont gets control it does further diagnosing on the read commands. The cache will not process the following types of reads:

READ COMMANDS NOT CACHED

1) Any Read Count, Key, data - selector
2) Read over BLOCK boundary
3) Read over track boundary
4) Multi-track reads
5) Reads with no data transfer (selector) set
6) Any read greater than 50% of BLOCK size (i.e., if block size is 12 and 10 records are requested)

Once a read command has passed these tests it is cached. The cache buffers are searched for a "hit" on the record(s) being read. If a hit is encountered, the data is moved to the users buffer area and the command is posted as completed with normal status. If a hit is not encountered, a block is read from the disk which contains the records requested. The appropriate data rocords are moved to the users buffer and his I/O is posted complete. The least recently used buffer is used to contain this data. This "miss" is a critical performance consideration because as the "miss" rate goes up, the disk I/O increases more than it would without cache. This buffer also becomes the most recently used and thereby would be the last of all current buffers to be purged. This would be true every time this buffer is access, this gives you an idea of how certain segments could reside for extended periods of time in cache.

All write data commands are manipulated by the cache symbiont. We process them in a method we call "write thru" which means we update a buffer in cache which has records being written. This write thru mechanism is critical to allow for the proper support of read caching. This says that to allow for reading blocks just written these blocks must be kept current in the cache as well as the disk. After this checking the write is issued. If an error occurs, then the cache buffer is purged.

The search read greater or equal commands are supported on the Model 4/6 on disks 8417/19 under R8.2. Only single track searches are allowable, and even if the buffer size is not enough to hold one track a separate buffer is used initially and then broken up accordingly. The track is read and searched and the data is returned or not accordingly.

This is a new function which during testing is showing a high statistics hit rate, we would appreciate all feedback on its effect on performance. Appropriate for applications which do excessive searches.

PERFORMANCE

The improvement from caching in throughput or response time is application dependent. Job and/or function execution times will generally decrease exponentially as the cache size increases. Batch users, which tend to process sequentially, will generally have a higher hit rate for a particular cache than interactive users which tend to process randomly. The reverse is also true. Therefore, one cache size will not optimise for both but one size will give the best overall system performance.

The performance you will receive from the OS/3 cache is hard to predict but easy to document. The use of varied cache buffer sizes while doing daily production will provide data to tell you which size works best. Use of the cache statistics (CMSTAT), job execution times and response times will guide you in your decision. Special emphasis should be given to sizes near the block size definition boundaries. These boundaries could show distinct differences in performance.

Another consideration is to vary your cache size with your shifts. Using a different size for day processing and another for night. Especially if more interactive processing is done during the day or vice versa.

By keeping logs for a few weeks or days you will be able to time your system performance.

There are many other techniques used to achieve performance gains on OS/3, these include modifying programs to use large block sizes, careful file placement to reduce seek time and adding extra channels, control units or disks.

However, these techniques require extensive and costly modifications to your system and may not be feasible. The OS/3 Disk Cache, on the other hand, provides an easy to use free mechanism to increase the performance of your System 80 without any changes to your system hardware or user programs/applications.

I hope my presentation has shed light on the OS/3 Disk Cache and that you will be able to use it to increase your productivity.

Thank you.