z/TPF ECB Heap Enhancements

Systems Control Program

Michael Shershin

2024 TPF Users Group Conference
May 5-8, New Orleans, LA

IBM Z

Disclaimer

Problem Statement

Use of ECB heap has been growing over the past several years and it will only continue to grow. It is a key piece of infrastructure that is used in application modernization efforts.

- The current implementation has overhead especially in the threaded environment.
- There are several tuning values that can be used in z/TPF to reduce the overhead of ECB heap usage. However, usage data that is provided in data reduction and ZMOWN is not sufficient to set some of the tuning values.

As-Is General Background

malloc() and MALOC HEAP=31BIT

- ➢ Get an ECB heap buffer in the 31-bit ECB heap area

malloc64() and MALOC HEAP=64BIT

- ➢ Get an ECB heap buffer in the 64-bit ECB heap area

calloc() and CALOC HEAP=31BIT

- ➢ Get an ECB heap buffer and initialize it to zero in the 31-bit ECB heap area

calloc64() and CALOC HEAP=64BIT

- ➢ Get an ECB heap buffer and initialize it to zero in the 64-bit ECB heap area

As-Is General Background

realloc() and RALOC HEAP=31BIT

- ➢ Reallocate an existing ECB heap buffer to a different size buffer in the 31-bit ECB heap area

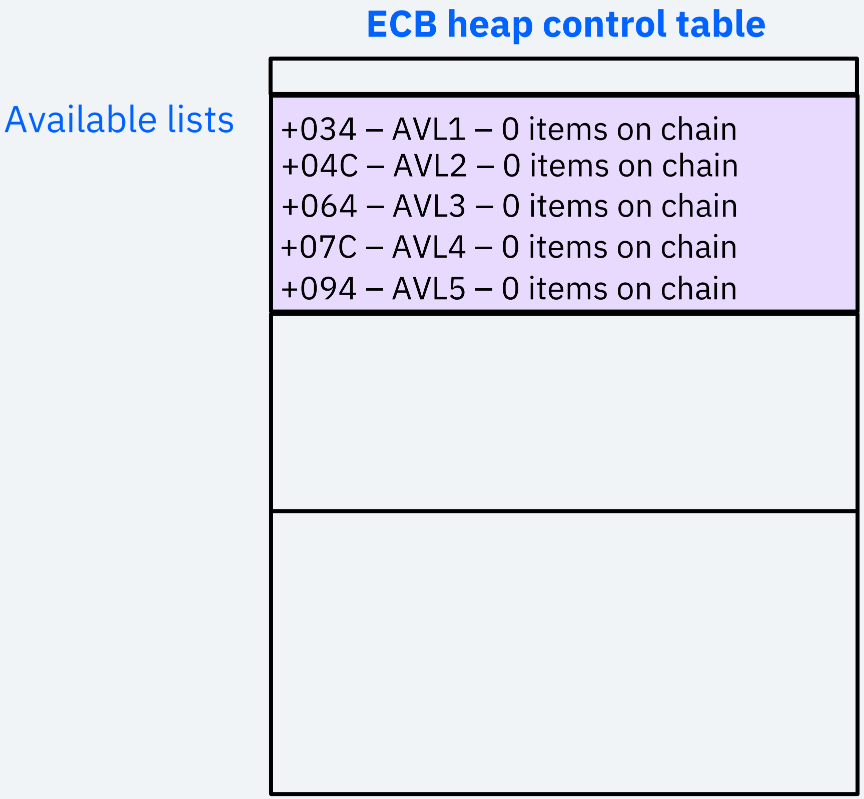
realloc64() and RALOC HEAP=64BIT

- ➢ Reallocate an existing ECB heap buffer to a different size buffer in the 64-bit ECB heap area

free() and FREEC

- ➢ Return an ECB heap buffer

As-Is General Background


ECB heap control table
Available lists
+034 – AVL1 – 0 items on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain

ECB heap control table

➢ Every ECB has its own ECB heap control table

Available lists: AVL1, AVL2, AVL3, AVL4, and AVL5

➢ Each AVL list contains pointers to a chain of ECB heap control entries that reference buffers that are available to be dispensed for use.

- Managed in a first-in first out (FIFO) order

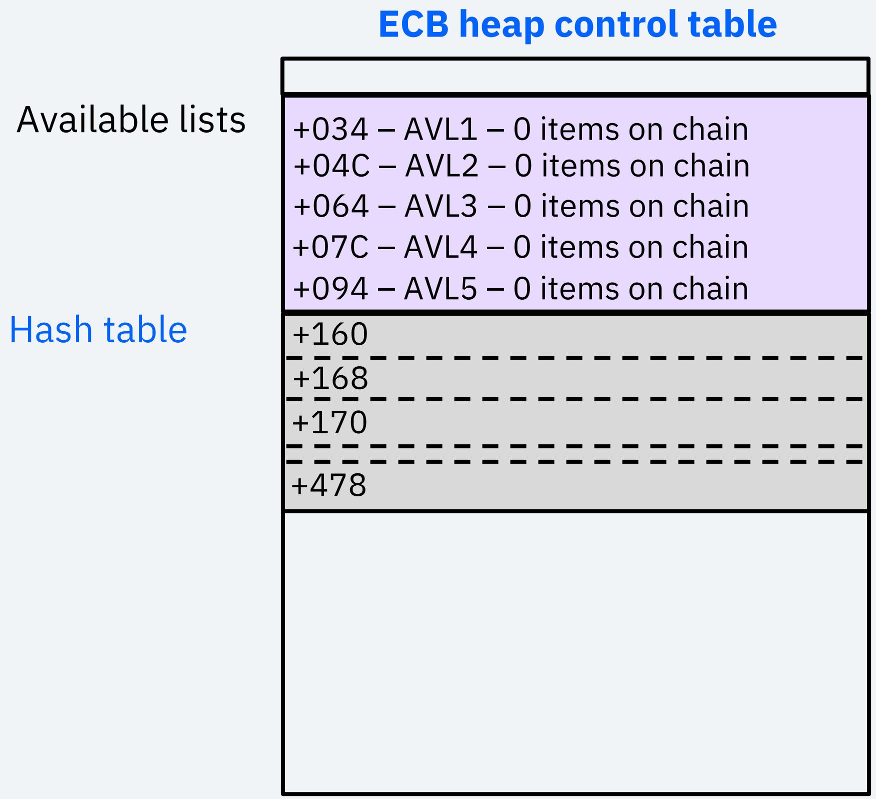
➢ AVL1, AVL2, AVL3, and AVL4 contain fixed size buffers

- Sizes of buffers on AVL1, AVL2, AVL3, and AVL4 are user configurable.

➢ AVL5 contains buffers of different sizes

➢ Only buffers on AVL5 are coalesced when a malloc() cannot find a buffer large enough to satisfy the request.

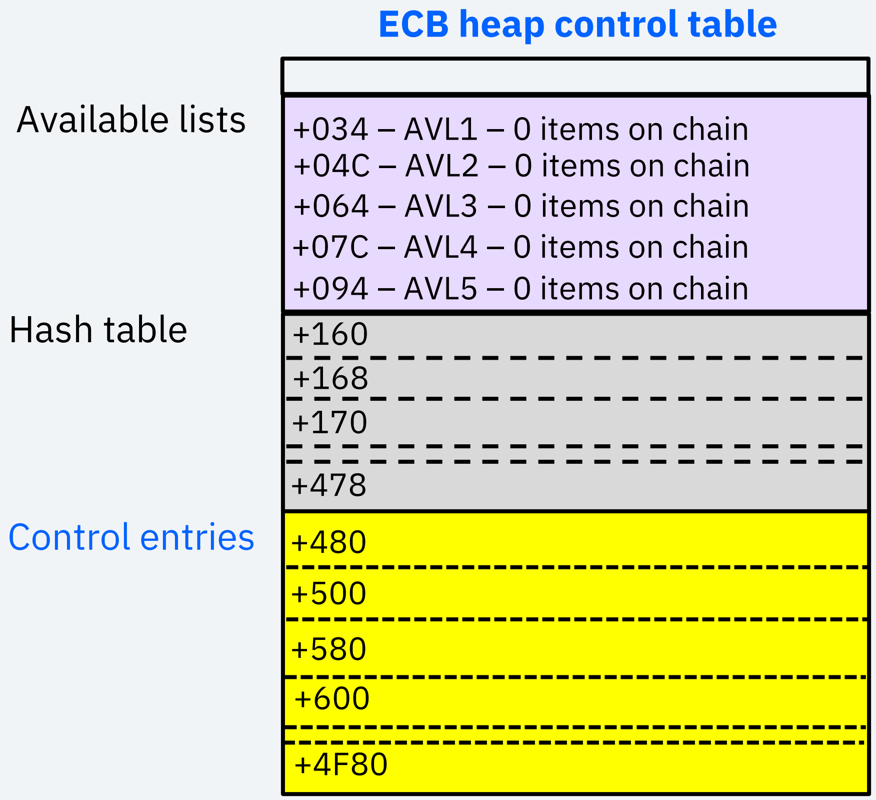
As-Is General Background


ECB heap control table
Available lists
+034 – AVL1 – 0 items on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Hash table
+160
+168
+170
+478

ECB heap hash table

- Initial allocation has 97 entries, but can be expanded dynamically
- ECB heap buffer address is used to locate a hash entry
- The hash entry points to the associated ECB heap control entry for an in-use heap buffer
- When the heap buffer is freed, the hash entry is cleared

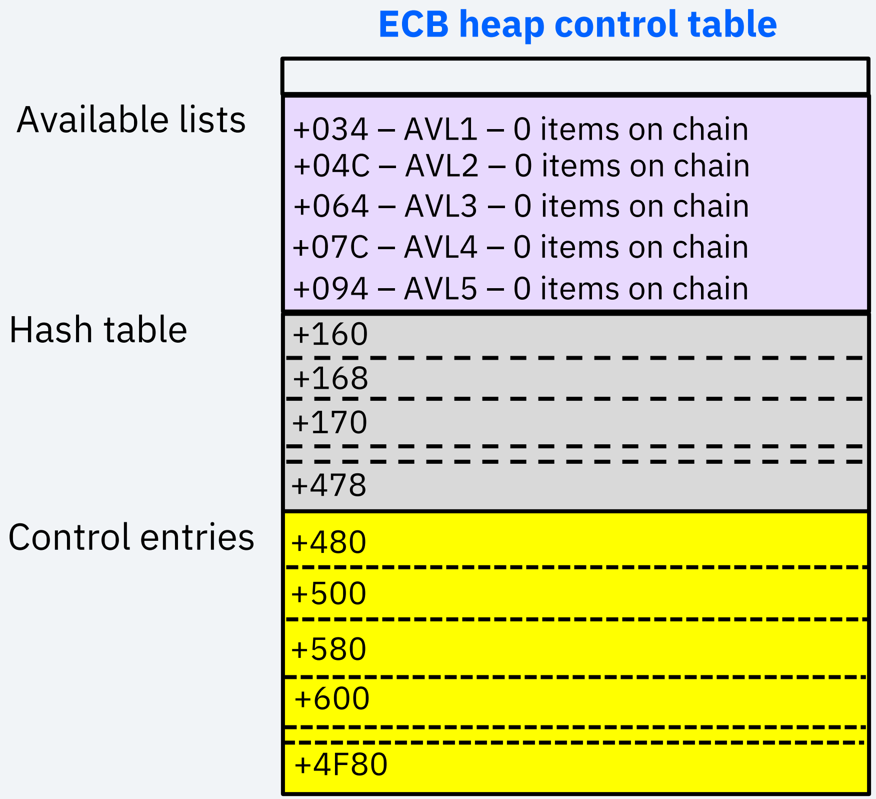
As-Is General Background


ECB heap control table
Available lists
+034 – AVL1 – 0 items on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Hash table
+160
+168
+170
+478
Control entries
+480
+500
+580
+600
+4F80

ECB heap control entries

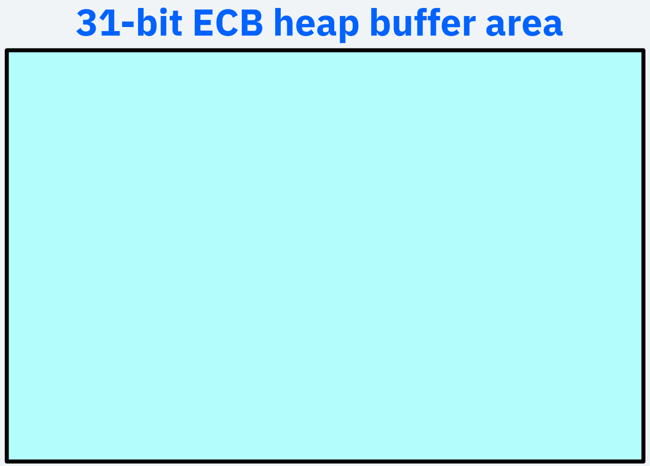
- Initially contains 151 entries but can be expanded dynamically
- Each entry contains information about an ECB heap buffer that is either in-use or on an available list

As-Is General Background

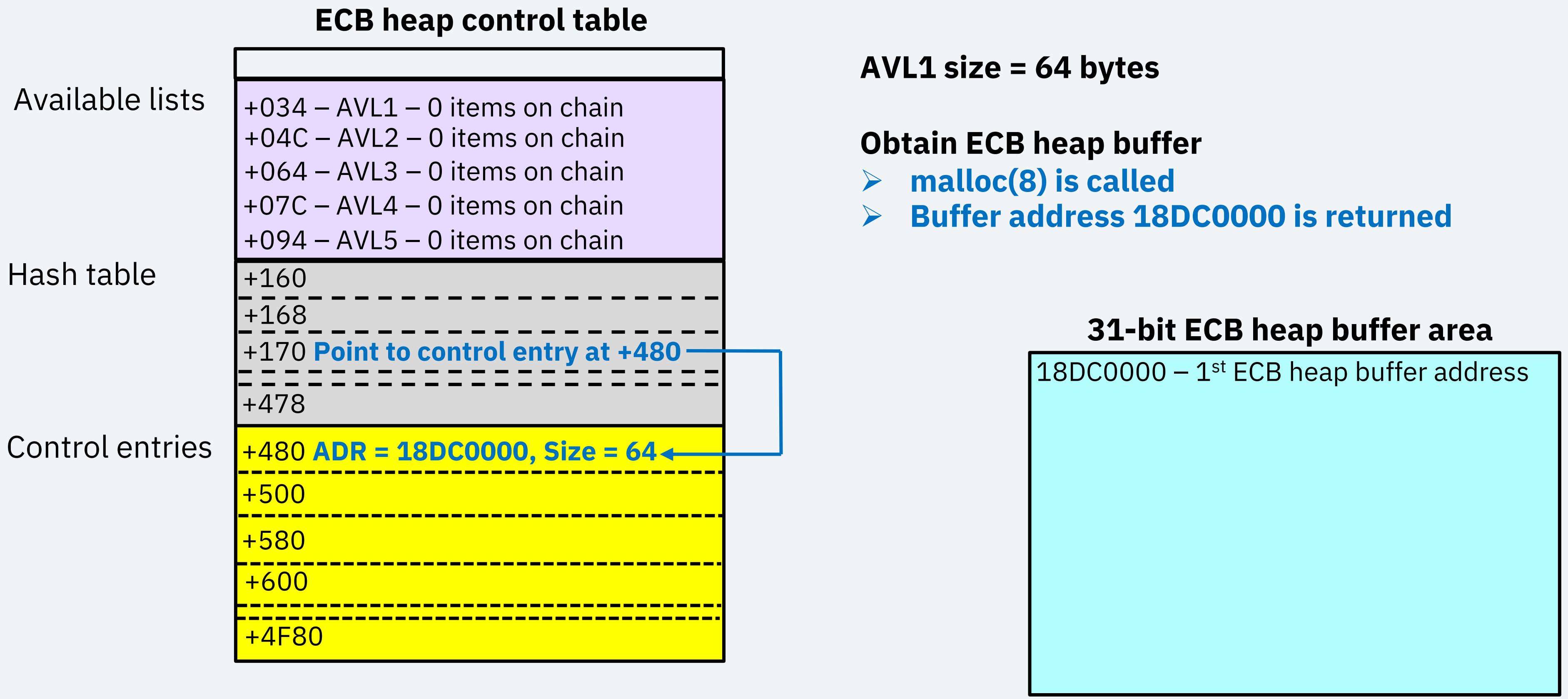

ECB heap control table
Available lists
+034 – AVL1 – 0 items on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Hash table
+160
+168
+170
+478
Control entries
+480
+500
+580
+600
+4F80

ECB heap buffer

- ➢ 31-bit buffer area (memory below 2 GB in the EVM)
- ➢ 64-bit buffer area (memory above 4 GB in the EVM)



As-Is General Background

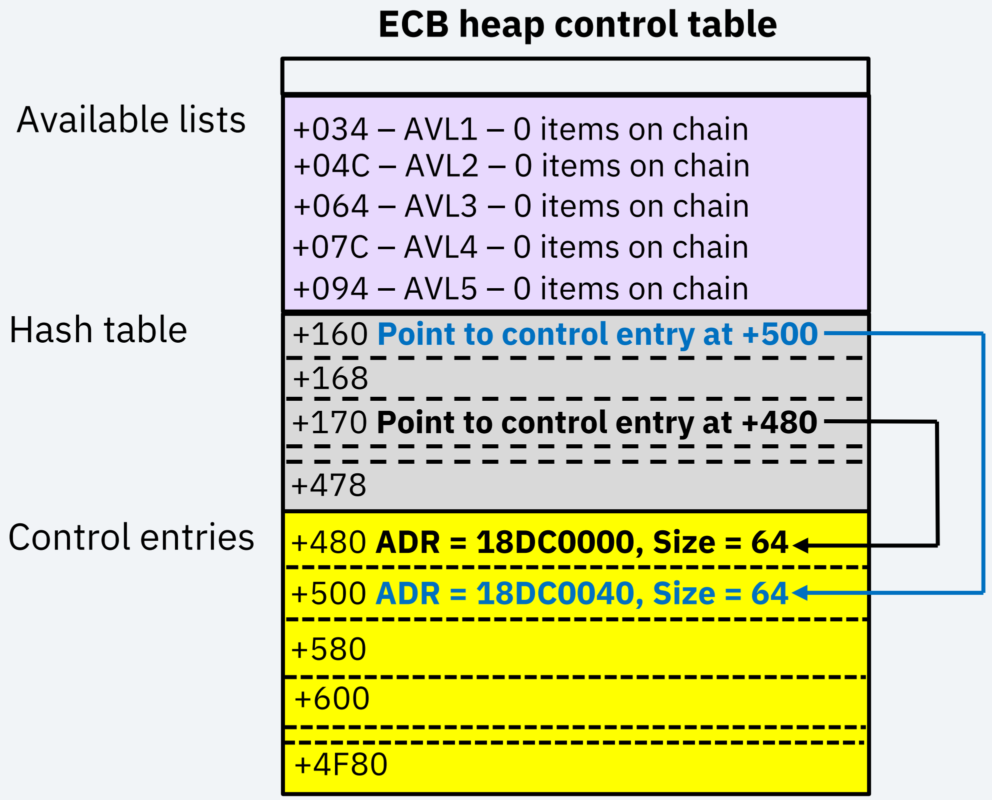

ECB heap control table
Available lists
+034 – AVL1 – 0 items on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Hash table
+160
+168
+170 Point to control entry at +480
+478
Control entries
+480 ADR = 18DC0000, Size = 64
+500
+580
+600
+4F80
31-bit ECB heap buffer area
18DC0000 – 1st ECB heap buffer address

AVL1 size = 64 bytes

Obtain ECB heap buffer

- **malloc(8) is called**
- **Buffer address 18DC0000 is returned**

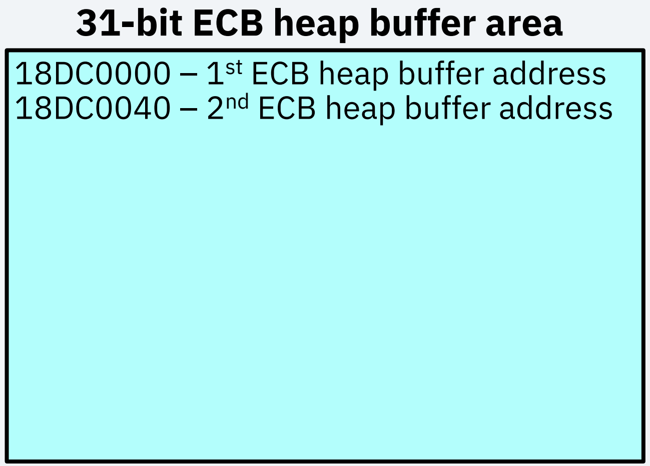
As-Is General Background


ECB heap control table
Available lists
+034 – AVL1 – 0 items on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Hash table
+160 Point to control entry at +500
+168
+170 Point to control entry at +480
+478
Control entries
+480 ADR = 18DC0000, Size = 64
+500 ADR = 18DC0040, Size = 64
+580
+600
+4F80

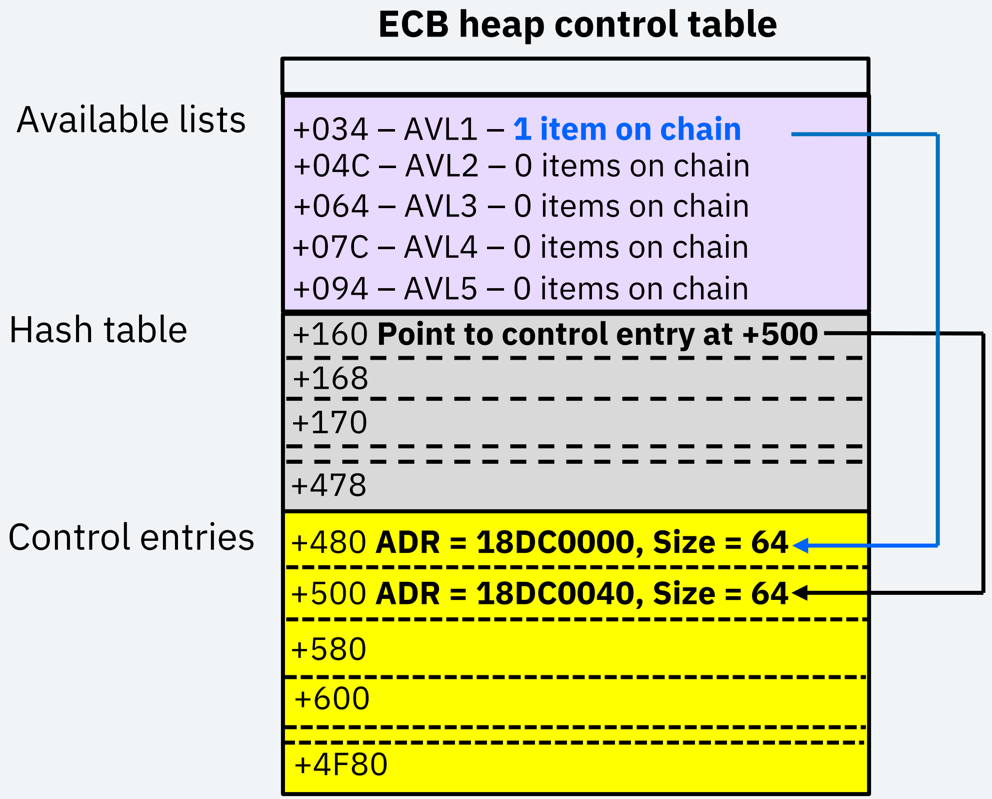
AVL1 size = 64 bytes

Obtain ECB heap buffer

- **malloc(32) is called**
- **Buffer address 18DC0040 is returned**


31-bit ECB heap buffer area
18DC0000 – 1st ECB heap buffer address
18DC0040 – 2nd ECB heap buffer address

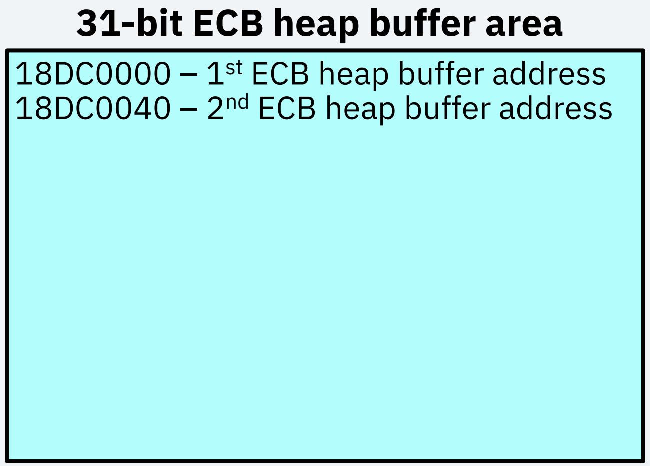
As-Is General Background


ECB heap control table
Available lists
+034 – AVL1 – 1 item on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Hash table
+160 Point to control entry at +500
+168
+170
+478
Control entries
+480 ADR = 18DC0000, Size = 64
+500 ADR = 18DC0040, Size = 64
+580
+600
+4F80

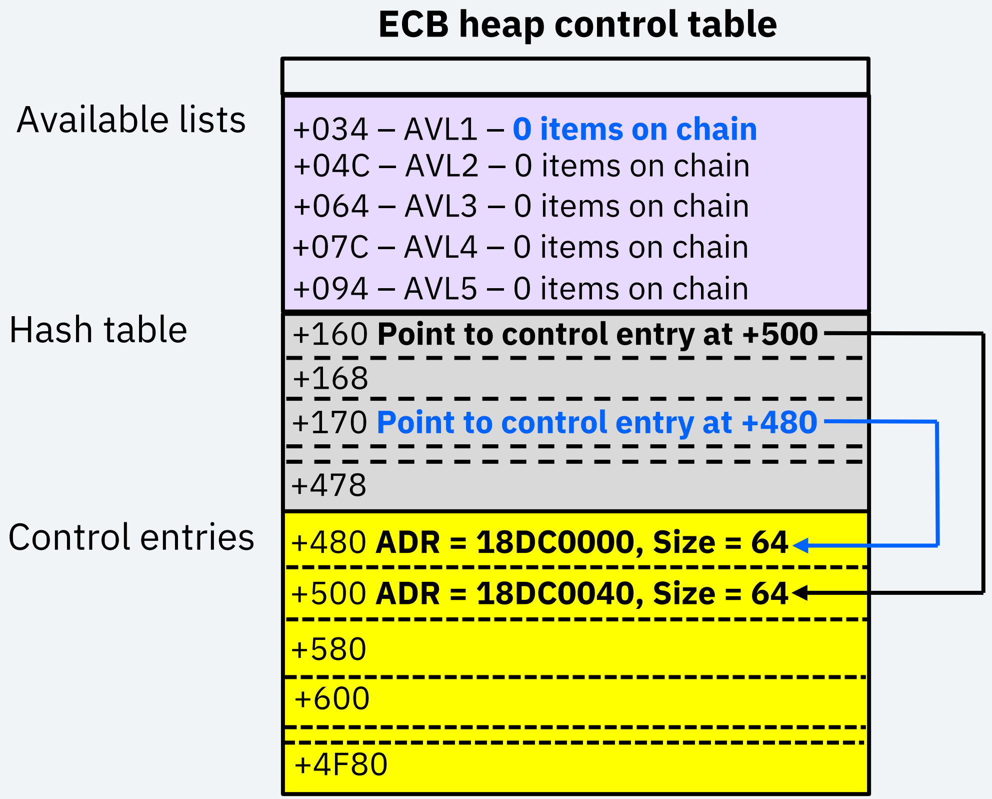
AVL1 size = 64 bytes

Free ECB heap buffer

➢ **free(18DC0000) is called**


31-bit ECB heap buffer area
18DC0000 – 1st ECB heap buffer address
18DC0040 – 2nd ECB heap buffer address

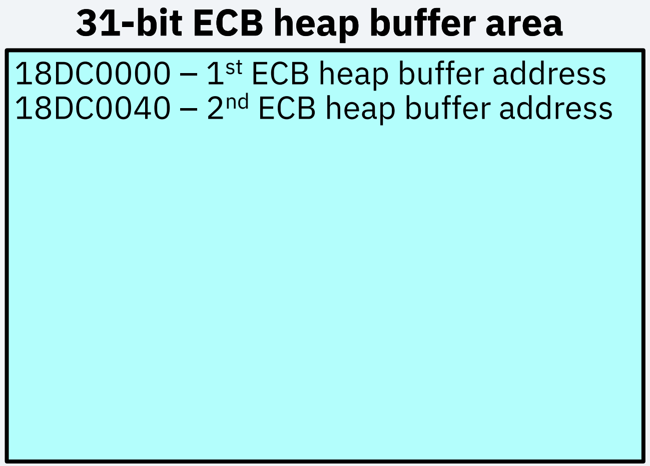
As-Is General Background


ECB heap control table
Available lists
+034 – AVL1 – 0 items on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Hash table
+160 Point to control entry at +500
+168
+170 Point to control entry at +480
+478
Control entries
+480 ADR = 18DC0000, Size = 64
+500 ADR = 18DC0040, Size = 64
+580
+600
+4F80

AVL1 size = 64 bytes

Obtain ECB heap buffer

- **malloc(48) is called**
- **Buffer address 18DC0000 is reused**


31-bit ECB heap buffer area
18DC0000 – 1st ECB heap buffer address
18DC0040 – 2nd ECB heap buffer address

ECB Heap Management

As-Is: ECB Heap Control Table Initialization

The ECB heap control table is used to manage ECB heap.

- ➢ The size of the ECB heap control table is x’5000’ (20,480) bytes.
- ➢ All x’5000’ bytes are initialized when the first malloc() call is done by the ECB.
- ➢ 151 ECB heap control entries are initialized.

An overflow area in system heap is obtained if more than 151 control entries are used.

- ➢ The size of one overflow area is x’8000’ (32,768) bytes and it contains 255 control entries.
- ➢ All 255 control entries are initialized when an overflow area is obtained.

Pain Points: ECB Heap Control Table Initialization

Overhead to initialize the ECB heap control table

- ➢ If an ECB does not use 151 control entries in the control table, initializing the unused control entries is wasteful.
  - Overhead to initialize unused control entries
  - Causes memory cache inefficiencies.
- ➢ If an overflow area is obtained and all 255 entries are not used, initializing the unused control entries is wasteful.

To-Be: ECB Heap Control Table Initialization

The following changes will be made:

- ➢ The entire ECB heap control table (x’5000’) bytes will NOT be initialized on first malloc() call.
  - The AVL lists and the hash table will be initialized.
  - The 151 ECB heap control entries will NOT be initialized.
  - Each ECB heap control entry will be initialized when it is first used.
- ➢ If an overflow area is needed for ECB heap control entries, the entire system heap buffer of x’8000’ bytes will NOT be initialized when the buffer is obtained.
  - Each ECB heap control entry will be initialized when it is first needed.

As-Is: AVL List Management

AVL1, AVL2, AVL3, and AVL4 lists contain buffers that are fixed sizes. The AVL5 list contains buffers that can be various sizes.

- ➢ For example, if the size of AVL1 buffers is 64 bytes, all buffers on AVL1 will be 64 bytes in size. If the size of AVL2 buffers is 512 bytes, all buffers on AVL2 will be 512 bytes.

Each AVL list contains pointers to a chain of ECB heap control entries for ECB heap buffers.

- ➢ Management of ECB heap control entries on AVL1, AVL2, AVL3, and AVL4 are done using a first-in first-out (FIFO) order.

Pain Points: AVL List Management

Overhead due to the use of FIFO ordering for AVL1, AVL2, AVL3, and AVL4

- Memory cache inefficiencies occur
  - When the next buffer is dispensed, FIFO ordering causes the buffer that has been on the AVL list the longest to be dispensed.
  - The buffer being dispensed might have aged out of memory cache because of the delay between usage.
  - The processor might incur added overhead to bring the cache line into memory cache.

To-Be: AVL List Management

The following change will be made:

- ➢ AVL1, AVL2, AVL3, and AVL4 lists will be managed in a last-in first-out (LIFO) order.
  - Reusing buffers quickly improves memory cache usage and improves performance.

As-Is: Coalescing of Available ECB Heap Buffers

Coalescing of ECB heap buffers

- ➢ Coalescing reduces buffer fragmentation by combining adjacent available buffers into a single larger buffer.
- ➢ Coalescing is only done when a malloc() cannot find an available buffer on AVL5.
- ➢ Coalescing is only done on buffers that are on the AVL5 list.
- ➢ Coalescing does not use buffers that are on the AVL1, AVL2, AVL3, or AVL4 lists.
  - When a buffer is allocated based on the size for AVL1, AVL2, AVL3, or AVL4, the buffer remains that size until the ECB exits.

Pain Points: Coalescing of Available ECB Heap Buffers

Fragmentation can happen in buffers on the AVL5 list

- ➢ Fragmentation in the ECB heap can result in a malloc() request not being able to obtain a buffer even though there is enough space. The problem is that the available space might not be in one continuous buffer.
  - Coalescing happens late only after a malloc() cannot find a buffer of the requested size on the AVL5 list.
  - As a result, buffers of varying sizes are obtained before coalescing is performed which increases the fragmentation in the ECB heap.

To-Be: Coalescing of Available ECB Heap Buffers

The following change will be made:

- ➢ Coalescing of buffers on the AVL5 list will happen when a free() is done on a buffer.
  - Coalescing is done earlier which maintains larger buffers on the AVL5 list.
  - Fragmentation within ECB heap is reduced, which increases the probability finding available space to satisfy a malloc() request.

ECB Heap in a Threaded Environment

As-Is: ECB Heap Hash Table Expansion

ECB heap hash table is typically expanded in a threaded environment.

- The hash table is in the ECB heap control table
- The hash size is 97 entries.
- If there are more than 971 ECB heap control entries in use, the hash table is expanded.
    - Searching the synonym chains when there are 97 hash table entries is too expensive
    - The expanded hash table is in a 1 MB system heap block.
    - The expanded hash table has 131,071 entries.
    - When the hash table is expanded, all entries in the original hash table are moved to the expanded hash table.

Pain Points: ECB Heap Hash Table Expansion

Overhead in a threaded environment (for example SSL and Java)

- Overhead exists to move 971 entries from the original hash table to the expanded hash table.

To-Be: ECB Heap Hash Table Expansion

The following changes will be made:

- ➢ The ECB heap hash table will be expanded when the threaded environment (process) is created. It will not wait for 971 entries to be in-use before expanding.
    - Fewer entries will need to be moved from the original hash table to the expanded hash table. Overhead will be reduced.

As-Is: Serialization

- The ECB heap control table is shared across all threads (ECBs) in a process.
- On every malloc(), calloc(), realloc(), and free() call the process lock is obtained to serialize access to the ECB heap control table across the threads in the process.
  - The storage protect key for the process lock is key F.
  - Because E-type code runs with key 1, every malloc(), calloc(), realloc(), and free() result in the equivalent assembler macro being called. The assembler macro issues an SVC and processes in the control program where the process lock is obtained.
  - In a nonthreaded environment processing for most malloc(), calloc(), and free() requests are done in the C function. The assembler macro is not called.

Pain Points: Serialization

Overhead in a threaded environment (for example SSL and Java)

- ➢ Every ECB heap API call incurs the overhead of an SVC call to obtain the process lock.

To-Be: Serialization

The following changes will be made:

- ➢ A new ECB heap lock will be created and used in place of the process lock.
  - The new lock will be in the ECB heap control table. The storage key is 1.
  - Standard E-type code that runs in PSW key 1 will be able to obtain the new lock. As a result, C function service routines for malloc(), calloc(), and free() will be able to obtain the lock.
    - An SVC call is no longer needed to obtain the new lock.
- ➢ Processing for most malloc(), calloc(), and free() will be done in the C function
  - Both threads and non-threaded ECBs will not need to do an SVC call.

As-Is: ECB Heap Trace

- A single ECB heap trace table is shared across all threads (ECBs) in a process.

Pain Points: ECB Heap Trace

Overhead in a threaded environment (for example SSL and Java)

- ➢ Memory cache inefficiencies happen because the cache lines for the ECB heap trace table are bounced between I-streams.
- ➢ Updates to the ECB heap trace table must be serialized across the threads in the process. The process lock is obtained to provide serialization.

To-Be: ECB Heap Trace

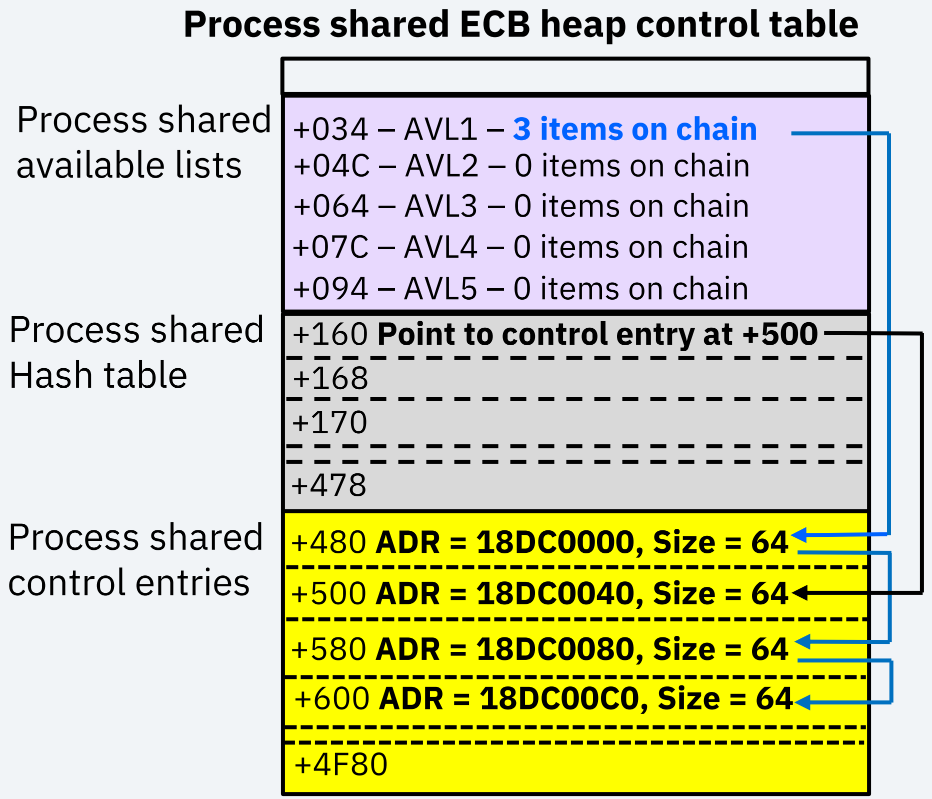
The following changes will be made:

- ECB heap trace table will be thread unique.
  - Eliminates bouncing of memory cache lines between I-streams.
  - Eliminates the need to obtain a lock to serialize updates to the table.

As-Is: AVL Lists for Threads

- The AVL lists in the ECB heap control table are process shared.
  - Because the AVL lists are process shared, an ECB heap buffer can be used by one thread on one I-stream; freed by the thread; and then obtained by a different thread on different I-stream.
  - The process lock is obtained to serialize updates to the ECB control table.

As-Is: AVL Lists for Threads


Process shared ECB heap control table
Process shared available lists
+034 – AVL1 – 3 items on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Process shared Hash table
+160 Point to control entry at +500
+168
+170
+478
Process shared control entries
+480 ADR = 18DC0000, Size = 64
+500 ADR = 18DC0040, Size = 64
+580 ADR = 18DC0080, Size = 64
+600 ADR = 18DC00C0, Size = 64
+4F80

Assume the following scenario

- **1 buffer is in-use**
- **3 buffers have been obtained and released and they are on the AVL1 list**

When a thread requests an ECB heap buffer, the process lock is obtained and process shared AVL1 lists are used.

31-bit ECB heap buffer area

18DC0000 – 1st ECB heap buffer address
18DC0040 – 2nd ECB heap buffer address
18DC0080 – 3rd ECB heap buffer address
18DC00C0 – 4th ECB heap buffer address

Pain Points: AVL Lists for Threads

Overhead in a threaded environment (for example SSL and Java)

- Reuse of a buffer on a different I-stream causes memory cache inefficiencies.
- Management of ECB heap control entries on the AVL lists requires serialization between the threads. A lock must be obtained to provide serialization.

To-Be: AVL Lists for Threads

Thread unique AVL1, AVL2, AVL3, and AVL4 lists will be created.

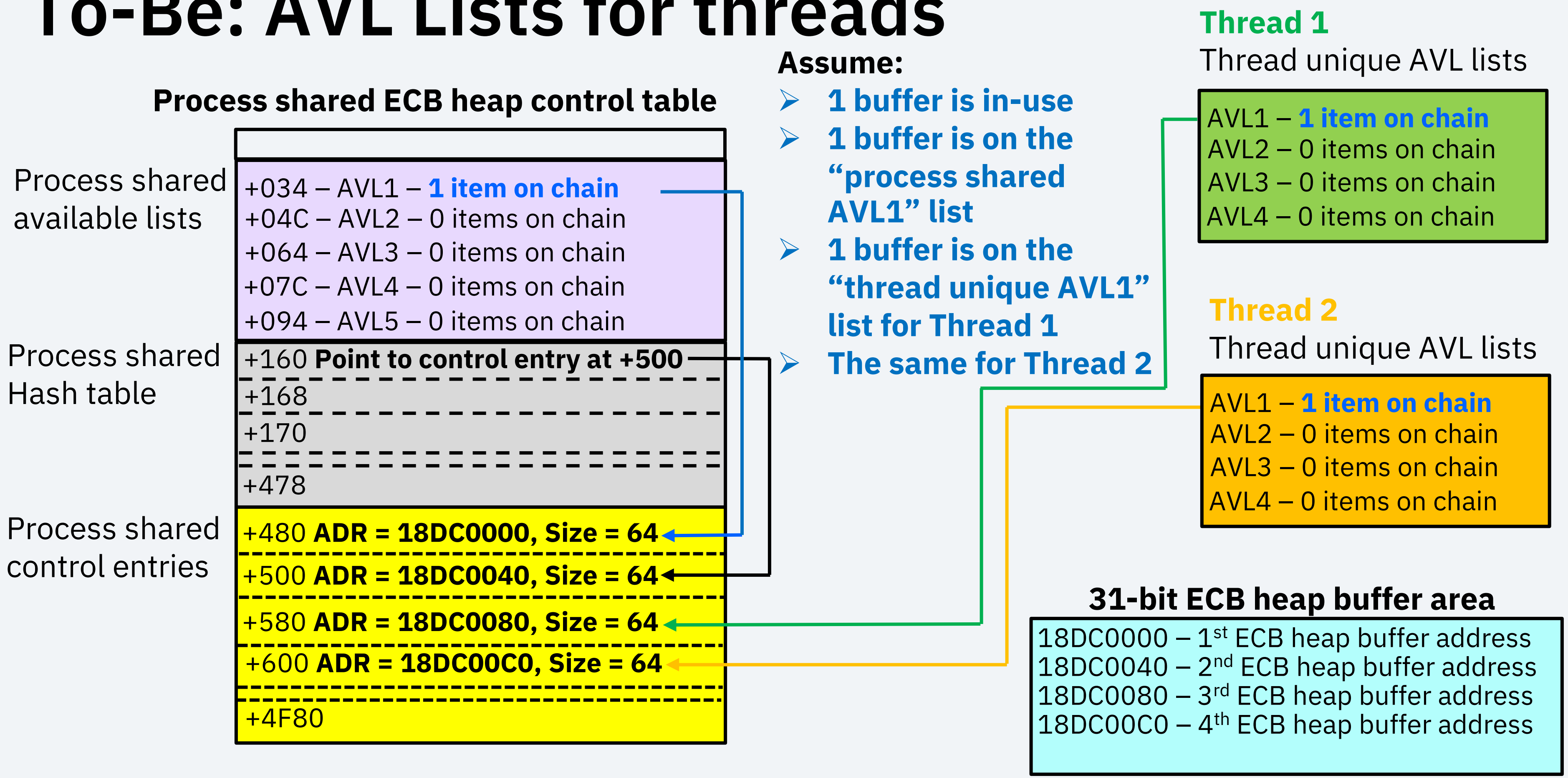
- ➢ AVL1, AVL2, AVL3 and AVL4 lists will have “thread unique lists” and “process shared lists”.
  - The AVL5 list will be “process shared” only.
- ➢ A lock will NOT be needed to manage the “thread unique AVL lists”.
- ➢ When a free() is done and the buffer is put on an AVL1, AVL2, AVL3, or AVL4 list, the processing steps are as follows:
  1. Add the ECB heap control entry to the “thread unique AVL list” if the number of buffers on the “thread unique AVL list” is less than a predefined limit.
  2. Otherwise, add the ECB heap control entry to the “process shared AVL list”. The ECB heap lock must be obtained to add to the “process shared AVL list”.

To-Be: AVL Lists for Threads

Thread unique AVL1, AVL2, AVL3, and AVL4 lists will be created.

- When a malloc() is done and the buffer is obtained from an AVL1, AVL2, AVL3, or AVL4 list, the processing steps are as follows:
    1. Check the appropriate “thread unique AVL list”. If a pointer to an ECB heap control entry is on the list, the first buffer will be used.
    2. If there are no entries on the appropriate “thread unique AVL list”, check the appropriate “process shared AVL list”. The ECB heap lock must be obtained before using the “process shared AVL list”.
    3. If there are no entries on the appropriate “process shared AVL list”, check the “process shared AVL5 list” for available buffers. The ECB heap lock must be obtained before using the “process shared AVL5 list”.
    4. Otherwise, return an error.

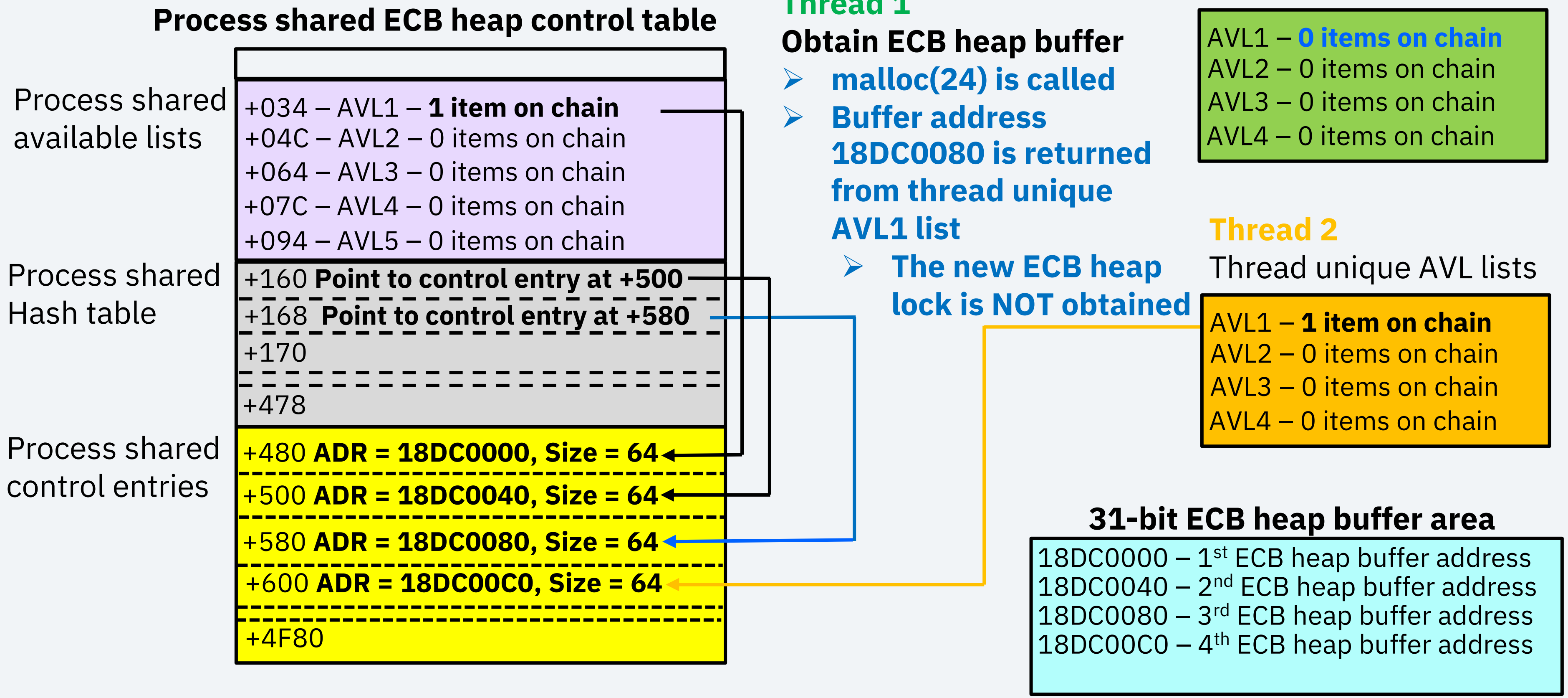
To-Be: AVL Lists for threads


Process shared ECB heap control table
Process shared available lists
+034 – AVL1 – 1 item on chain
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Process shared Hash table
+160 Point to control entry at +500
+168
+170
+478
Process shared control entries
+480 ADR = 18DC0000, Size = 64
+500 ADR = 18DC0040, Size = 64
+580 ADR = 18DC0080, Size = 64
+600 ADR = 18DC00C0, Size = 64
+4F80
Thread 1
Thread unique AVL lists
AVL1 – 1 item on chain
AVL2 – 0 items on chain
AVL3 – 0 items on chain
AVL4 – 0 items on chain
Thread 2
Thread unique AVL lists
AVL1 – 1 item on chain
AVL2 – 0 items on chain
AVL3 – 0 items on chain
AVL4 – 0 items on chain
31-bit ECB heap buffer area
18DC0000 – 1st ECB heap buffer address
18DC0040 – 2nd ECB heap buffer address
18DC0080 – 3rd ECB heap buffer address
18DC00C0 – 4th ECB heap buffer address

Assume:

- **1 buffer is in-use**
- **1 buffer is on the "process shared AVL1" list**
- **1 buffer is on the "thread unique AVL1" list for Thread 1**
- **The same for Thread 2**

To-Be: AVL Lists for Threads



Process shared ECB heap control table
Process shared available lists
+034 – AVL1 – **1 item on chain**
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain
Process shared Hash table
+160 **Point to control entry at +500**
+168 **Point to control entry at +580**
+170
+478
Process shared control entries
+480 **ADR = 18DC0000, Size = 64**
+500 **ADR = 18DC0040, Size = 64**
+580 **ADR = 18DC0080, Size = 64**
+600 **ADR = 18DC00C0, Size = 64**
+4F80
Thread 1
Thread unique AVL lists
AVL1 – **0 items on chain**
AVL2 – 0 items on chain
AVL3 – 0 items on chain
AVL4 – 0 items on chain
Thread 2
Thread unique AVL lists
AVL1 – **1 item on chain**
AVL2 – 0 items on chain
AVL3 – 0 items on chain
AVL4 – 0 items on chain
31-bit ECB heap buffer area
18DC0000 – 1st ECB heap buffer address
18DC0040 – 2nd ECB heap buffer address
18DC0080 – 3rd ECB heap buffer address
18DC00C0 – 4th ECB heap buffer address

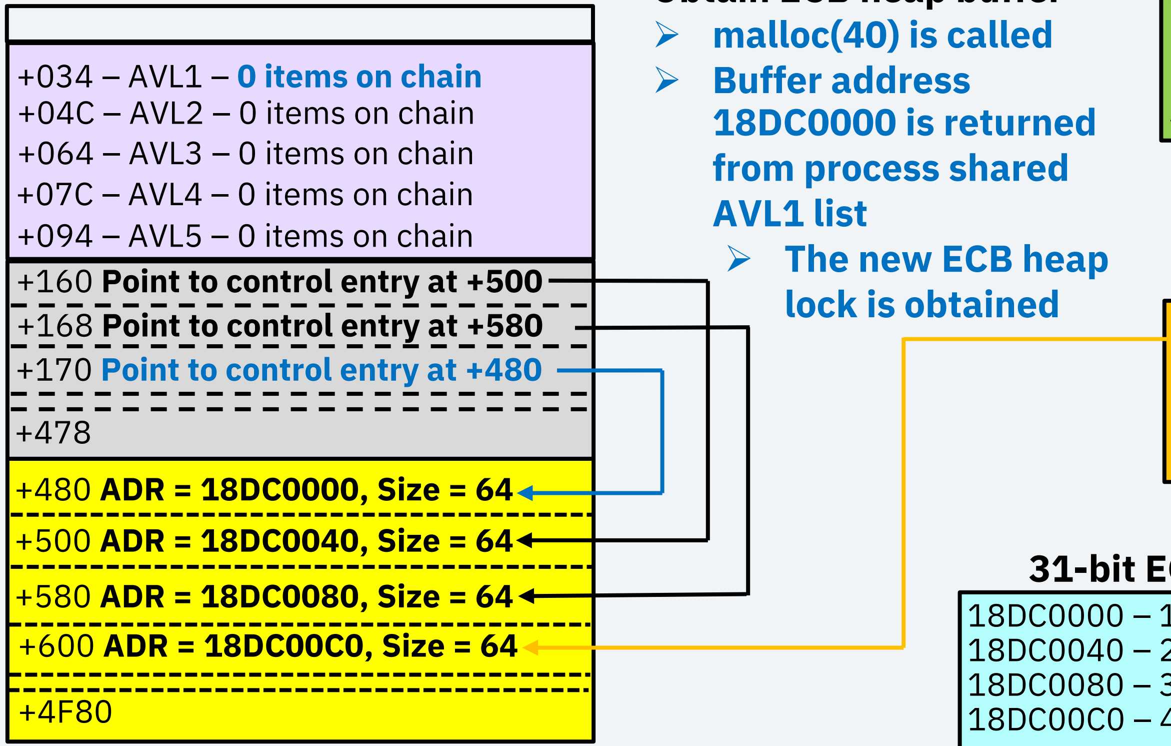
Thread 1

Obtain ECB heap buffer

- **malloc(24) is called**
- **Buffer address 18DC0080 is returned from thread unique AVL1 list**
  - **The new ECB heap lock is NOT obtained**

To-Be: AVL Lists for Threads

Process shared ECB heap control table



Process shared available lists

+034 – AVL1 – **0 items on chain**
+04C – AVL2 – 0 items on chain
+064 – AVL3 – 0 items on chain
+07C – AVL4 – 0 items on chain
+094 – AVL5 – 0 items on chain

Process shared Hash table

+160 **Point to control entry at +500**
+168 **Point to control entry at +580**
+170 **Point to control entry at +480**
+478

Process shared control entries

+480 **ADR = 18DC0000, Size = 64**
+500 **ADR = 18DC0040, Size = 64**
+580 **ADR = 18DC0080, Size = 64**
+600 **ADR = 18DC00C0, Size = 64**
+4F80

Thread 1
Obtain ECB heap buffer

- **malloc(40) is called**
- **Buffer address 18DC0000 is returned from process shared AVL1 list**
  - **The new ECB heap lock is obtained**

Thread 1
Thread unique AVL lists

AVL1 – **0 items on chain**
AVL2 – 0 items on chain
AVL3 – 0 items on chain
AVL4 – 0 items on chain

Thread 2
Thread unique AVL lists

AVL1 – **1 item on chain**
AVL2 – 0 items on chain
AVL3 – 0 items on chain
AVL4 – 0 items on chain

31-bit ECB heap buffer area

18DC0000 – 1st ECB heap buffer address
18DC0040 – 2nd ECB heap buffer address
18DC0080 – 3rd ECB heap buffer address
18DC00C0 – 4th ECB heap buffer address

ECB Heap Tuning

As-Is ECB Heap Tuning Values

The following are ECB heap values that can be set:

- ➢ Size of the preallocated ECB heap area, which is used for 31-bit ECB heap.
- ➢ Size of buffers on the AVL1, AVL2, AVL3, and AVL4 lists.
- ➢ Number of preallocated buffers on the AVL1, AVL2, AVL3, and AVL4 lists.
- ➢ Maximum size of the virtual area for both 31-bit and 64-bit ECB heap.
- ➢ Maximum amount or memory that an ECB can obtain for both 31-bit and 64-bit ECB heap.

As-Is ECB Heap Data Collection

The following are data collection reports that are used to set the ECB heap tuning values:

- ➢ Data reduction: ECB HEAP STATISTICS
  - Used to set the number of preallocated buffers on the AVL1, AVL2, AVL3, and AVL4 lists.
- ➢ Data reduction: TPF ECB 31-BIT HEAP AREA USAGE REPORT
  - Used to understand the usage of 31-bit ECB heap
- ➢ Data reduction: TPF ECB 64-BIT HEAP AREA USAGE REPORT
  - Used to understand the usage of 64-bit ECB heap

As-Is ECB Heap Data Collection

The following are data collection reports that are used to set the ECB heap tuning values:

- ➢ Data reduction: TPF 31-BIT PREALLOCATED ECB HEAP STORAGE USAGE REPORT
  - Used to understand the usage of the preallocated ECB heap area.
- ➢ Data reduction: TPF ECB HEAP REQUEST SIZE REPORT
  - Used to set the buffer sizes for the AVL1, AVL2, AVL3, and AVL4 lists.

As-Is ECB Heap Data Collection

The following are data collection reports that are used to understand ECB heap usage within an application:

- ZMOWN: Number of ECB heap buffers obtained by ECB owner name
- ZMOWN: Number of ECB control entries obtained by ECB owner name.
- Name-value pair collection: Number of ECB heap buffers obtained by name-value pair
- Name-value pair collection: Number of ECB control entries obtained by name-value pair

Pain Points

Data collection and reporting lacks information.

- ➢ There is no report to help determine how large to make the preallocated ECB heap area.
- ➢ The TPF ECB HEAP REQUEST SIZE REPORT is not granular enough to determine how to set the buffer sizes for AVL1, AVL2, AVL3, and AVL4 lists.
  - The smallest size in the report is 128 bytes.
  - A large percentage of ECB heap buffers are 128 bytes or smaller.
- ➢ Data provided by ZMOWN and name-value pair collection is very high level. It does not provide enough detail to understand application behavior.
- ➢ Do not know how often the ECB heap hash table is expanded.

To-Be ECB Heap Data Collection

The following changes will be made:

- The TPF ECB HEAP REQUEST SIZE REPORT will be changed to use smaller size intervals.
- A new report called TPF 31-BIT VIRTUAL ECB HEAP STORAGE USAGE REPORT will be created.
  - Show usage of the 31-bit virtual area in 4 KB sizes.
  - Intended to better size 31-bit preallocated ECB heap

To-Be ECB Heap Data Collection

ZMOWN and name-value pair collection will provide the following information:

- Number of ECBs that obtain at least one ECB heap buffer.
- Number of ECBs that expand the ECB heap hash table.
- Number of ECBs that obtain at least one 1 MB frame ECB heap for both 31-bit and 64-bit ECB heap.
- Number of 1 MB frames obtained for both the 31-bit and 64-bit ECB heap.
- Number of AVL1, AVL2, AVL3, AVL4, and AVL5 buffers obtained.
- Number of AVL1, AVL2, AVL3, AVL4, and AVL5 buffers that were freed.
- High-water mark for usage in both 31-bit and 64-bit ECB heap.

To-Be ECB Heap Tuning Value

The following tuning value will be removed:

- ➢ Number of preallocated buffers on the AVL1, AVL2, AVL3, and AVL4 lists.
  - Most AVL lists have zero, one, or two preallocated buffers. Experience has shown that a small number of preallocated buffers provides better performance.
  - Changes in how the ECB heap control table is initialized need to have no preallocated AVL buffers.

Value Statement

- Improved performance when using ECB heap
  - For all ECB heap users
    - Use LIFO management of buffers on the AVL1, AVL2, AVL3, and AVL4 lists
  - For threaded environments
    - Thread unique AVL1, AVL2, AVL3, and AVL4 lists
    - Thread unique trace table
    - New ECB heap lock
- Improved tuning capabilities
- Better management of the ECB heap buffer area
  - Less fragmentation

Target Delivery Date

The target delivery date is 4Q2024.

Be a sponsor user

Sponsor users assist in design and implementation, and your feedback drives our development cycle.

Target personas

- Systems developers
- Performance analyst

Begins

On going

Interested? Contact

Michael Shershin e-mail: shershin@us.ibm.com

Thank you