ECB Heap Performance Enhancements (PJ47265)

Systems Control Program

Zach Polleck

2025 TPF Users Group Conference
May 5-7, Austin, TX

IBM Z

Background

ECB heap is contiguous ECB virtual memory (EVM) storage that can be allocated in buffers of arbitrary size.

The following C functions and assembler macros are used to manage ECB heap buffers:

- malloc, calloc, realloc, free (Standard C functions)
- malloc64, calloc64, realloc64 (TPF C functions)
- MALOC, CALOC, RALOC, FREEC (TPF HLASM macros)

Problem Statement

As usage of ECB heap continues to grow because of application modernization, the performance of ECB heap buffer management is increasingly critical.

This is true for applications that use ECB heap directly as well as system services that make heavy use of ECB heap such as shared SSL and Java.

Solution

The ECB heap buffer management routines have been updated to improve performance, especially in threaded environments.

Apply APAR PJ47265 (Dec 2024) and APAR PJ48131 (Jan 2025)

No application changes are needed to take advantage of this support.

Technical Details

PJ47265 introduces many performance enhancements, including:

- Initialize ECB heap control entries only when needed
- Use a hash table for locating tagged ECB heap buffers
- Align buffers on cache line boundaries
- Use LIFO dispatching for AVLs
- Use a new lock to avoid costly SVCs
- Make ECB heap trace table thread unique
- New thread unique AVLs to reduce lock contention for the shared AVLs.
- ECB HEAP REQUEST SIZE REPORT improvements to help tune AVL sizes.

For more technical details, see the ECB heap presentation from the 2024 TPFUG.

Performance Improvements: Threads

In the IBM lab, one test of the ECB heap performance enhancements was a driver that exchanged messages over Transport Layer Security (TLS) using z/TPF shared SSL support. With PJ47265 applied, we saw more than a 10% reduction in milliseconds per message.

	Mean message rate	**Mean milliseconds per message**
Baseline	14772/sec	0.068
With PJ47265	16603/sec	0.060

Shared SSL runs in a threaded environment with significant ECB heap usage.

Note that the driver contained no business logic to process messages. Your mileage might vary.

Performance Improvements: Locate ECB Heap

ECB heap supports the ability to tag certain buffers with names:

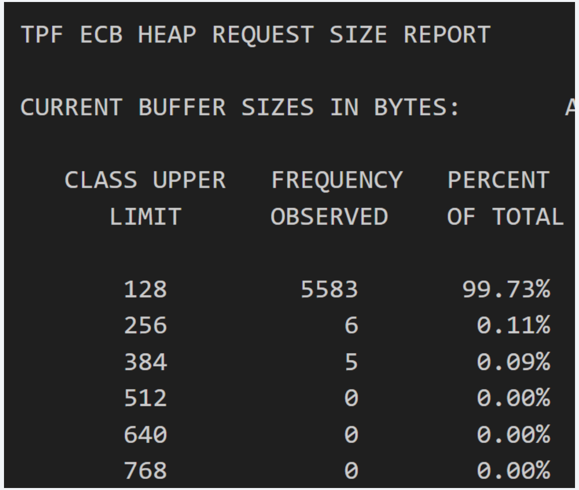
- Tag an ECB heap buffer: tpf_eheap_tag, EHEAPC FUNC=CREATE
- Locate a tagged buffer: tpf_eheap_locate, tpf_heap_locate_ext, EHEAPC FUNC=LOCATE

Previously, a costly sequential search was needed to locate buffers. In PJ47265, a hash table was added and locating tagged ECB heap buffers is now much faster.

Tuning Improvements

The ECB HEAP REQUEST SIZE REPORT has been changed to provide a wider range of bucket sizes which allows you to more easily tune size values for your ECB heap available lists.

Previous



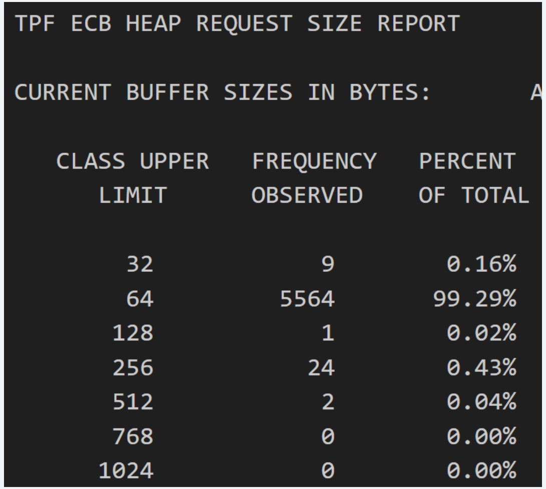
```
TPF ECB HEAP REQUEST SIZE REPORT

CURRENT BUFFER SIZES IN BYTES:        A

   CLASS UPPER    FREQUENCY    PERCENT
      LIMIT        OBSERVED    OF TOTAL

       128           5583       99.73%
       256              6        0.11%
       384              5        0.09%
       512              0        0.00%
       640              0        0.00%
       768              0        0.00%
```

PJ47265



```
TPF ECB HEAP REQUEST SIZE REPORT

CURRENT BUFFER SIZES IN BYTES:       A

   CLASS UPPER    FREQUENCY    PERCENT
      LIMIT        OBSERVED    OF TOTAL

        32              9       0.16%
        64           5564      99.29%
       128              1       0.02%
       256             24       0.43%
       512              2       0.04%
       768              0       0.00%
      1024              0       0.00%
```

Disclaimer

Any reference to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

Remaining Pain Points

Additional data needs to be collected to better understand ECB heap usage and tune ECB heap definitions.

- PJ47265 added better reporting to help determine appropriate values for ECB heap available list sizes.
- Additional data is needed to know how large the preallocated ECB heap area should be.
- Additional data would help better understand overall ECB heap usage.

Shared SSL and Java contain ported code and run in long-running ECBs. Therefore, they likely use ECB heap differently from your applications.

To-Be: Preallocated Heap Tuning

A new TPF 31-BIT VIRTUAL ECB HEAP STORAGE USAGE REPORT will be created.

- This report will show usage of the 31-bit virtual area in 4 KB increments.
- This report will be useful when tuning your preallocated ECB heap size.

To-Be: Understanding ECB Heap Usage

ZMOWN and name-value pair collection will have 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 times AVL1, AVL2, AVL3, AVL4, and AVL5 buffers were dispensed.
- Number of times AVL1, AVL2, AVL3, AVL4, and AVL5 buffers were freed.
- High-water mark for usage in both 31-bit and 64-bit ECB heap.

To-Be: Shared SSL and Java

The size of the buffers on the ECB heap available lists will be separately tunable for shared SSL and Java ECBs.

- Shared SSL and Java tend to use a significant number of small buffers, and therefore benefit from small ECB heap available list sizes.
- Your applications are likely to benefit from larger available list sizes.
- In the future, new system services with unique heap usage patterns may make use of separate available list sizes.

Target Delivery Date

The target delivery date is 3Q2025.

Be a sponsor user

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

Target personas

- Systems developers
- Performance analysts

Begins

Ongoing

Interested? Contact

Michael Shershin (shershin@us.ibm.com)

Thank you