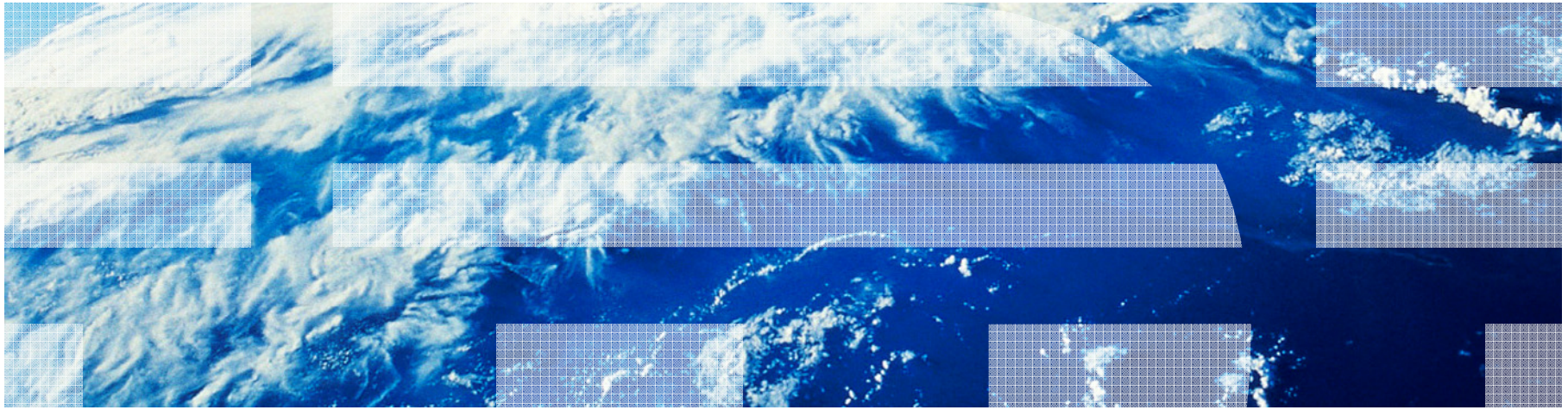
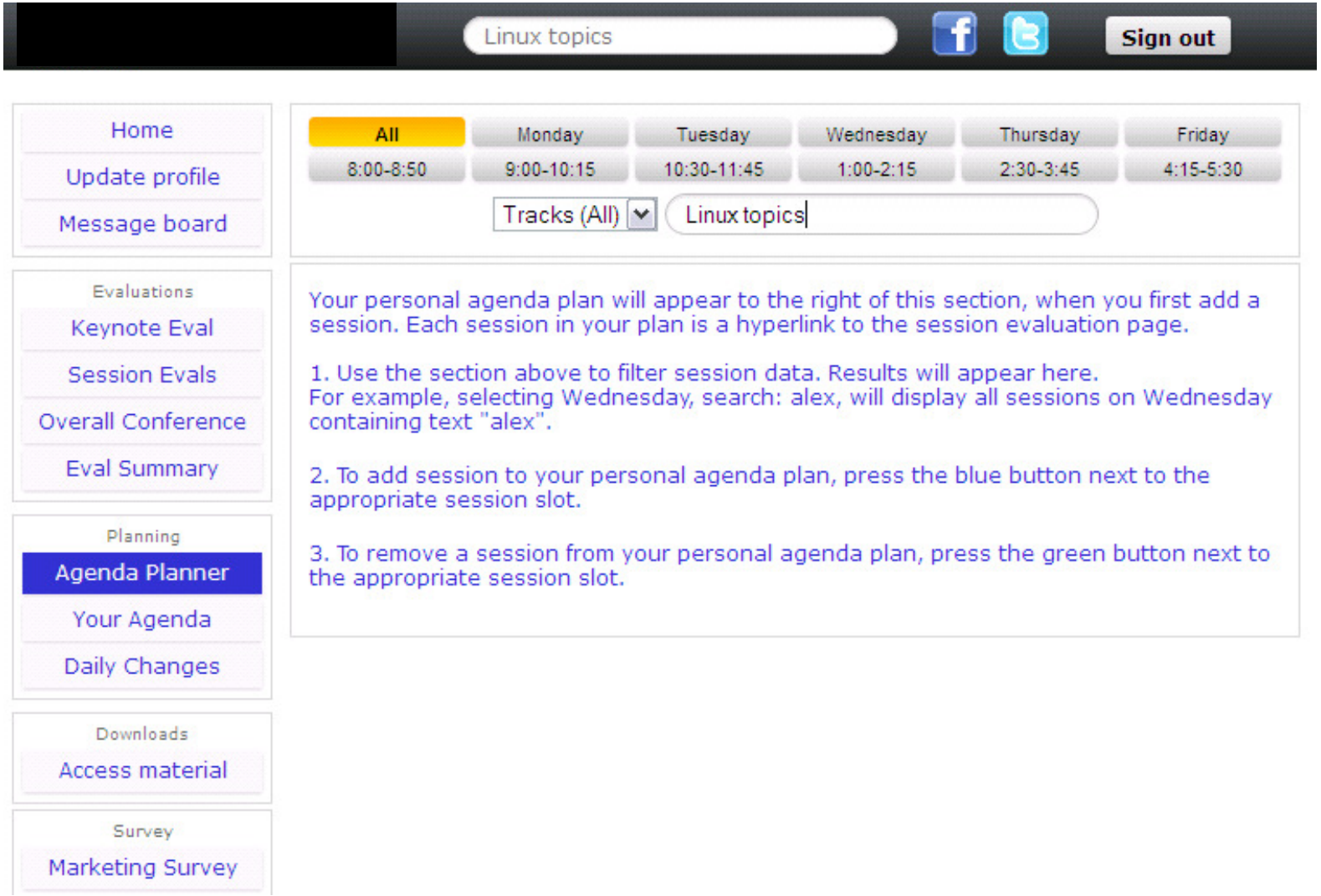
ST08
Anatomy of a memory leak

Grover Davidson – Development Support



IBMTECHU.COM



IBMTECHU.COM

- **IBM STG Technical Universities & Conferences web portal**
- **Direct link: ibmtechu.com/uk**
- **KEY FEATURES...**
  - Create a personal agenda using the agenda planner
  - View the agenda and agenda changes
  - Use the agenda search to find the sessions and/or
  - Download presentations
  - **Submit Session and Conference Evaluations**

Agenda

- **Define ‘memory leak’**
- **How is a ‘memory leak’ seen**
- **Details of memory assignment and growth**
- **Using debug malloc**
- **Case study 1 – Design**
- **Case study 2 – Producer/consumer model**
- **Case study 3 – We don’t do that**
- **Case study 4 – Failed to maintain**

Define ‘memory leak’

- **Term used to denote unexpected memory usage by a program**
- **Can be caused by many things including:**
  - Application design
  - Race conditions
  - Developer error
- **Generally challenging to track down**
- **Frequently requires source code to understand**

Monitoring memory usage

- **Remember that malloc does NOT increase the memory in use by an application**
- **Malloc does increase the addressable memory space**
- **Accessing a virtual address the first time results in real memory being assigned to the segment**
- **svmon, ps and other tools may be used for monitoring memory**
- **svmon gives the most detailed information**

Svmon example

```
# svmon –P 520224
 Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
  520224 test4            19290     8121        0    19283      N     Y     N
    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
    eb09d        d work shared library text         s   7170     0    0  7170
    a9cb7        2 work process private             s     48     3    0    48
    61cee        f work shared library data         s     40     0    0    40
    29ce7        - work                             s     10     3    0    10
    19d01        1 clnt code,/dev/hd1:128676        s      7     0    -     -

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
  520224 test4            19652     8121        0    19645      N     Y     N
    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
    eb09d        d work shared library text         s   7259     0    0  7259
    a9cb7        2 work process private             s    311     3    0   311
    61cee        f work shared library data         s     40     0    0    40
    29ce7        - work                             s     10     3    0    10
    19d01        1 clnt code,/dev/hd1:128676        s      7     0    -     -
```

- Note the *change* in the Inuse count for segment 2. This segment has had 263 additional pages accessed between these 2 samples of the svmon command for PID 520224.
- This clearly shows a growth in the size of this segment but gives no answers as to why it grew
- Many customers would consider this clear evidence of a memory leak
- The virtual column shows to total virtual pages accessed for a segment

Details of malloc and inuse

- **Mallocing memory only expands the valid addressable memory space in a segment**
- **A memory frame is assigned durring first accesses a virtual address**
- **Malloc usually touches the last page assigned and may cause a physical frame to be assigned**
- **The memory frame can be ‘stolen’ by LRU and will then appear in the Pgsp statistics of svmon**
- **The paging space may not be released when a new frame is assigned to the virtual address – this is to say both a memory frame and paging space page may be assigned to the same virtual address at the same time**
- **Freeing a memory address does NOT remove the association between the memory frame/paging space and the virtual address**

Debug Malloc

- **Does not require any changes to source code**
- **Libraries are shipped by default with AIX**
- **Creates a large amount of highly detailed data for every malloc that is not freed**
- **Provides additional functionality to debug memory problems**

Enabling debug malloc

- **Ensure you have the latest bos.rte.libc installed**
- **Setup the environment:**

```
$ export LANG=C
$ export MALLOCTYPE=debug
$ export MALLOCDEBUG=output:/tmp/malloc.debug,stack_depth:20, \
   report_allocations,log:extended
$ ./test4
<test4 terminates>
```

- **Output is not generated until the process terminates**
- **Only mallocs that are not freed are recorded by default**
- **Examine the output file from a **different** shell:**

  Exporting MALLOCTYPE/MALLOCDEBUG will affect other commands run from this shell

The gory details

- **Each malloc that is not explicitly freed will have a record in the output file:**

```
Allocation #15: 0x3060DFF8
    Allocation size: 0x4
    Allocated from heap: 0
    Allocation traceback:
    0xD01CA874  malloc
    0x10010700  load_history
    0x10000B84  main
    0x1000016C  __start
```

This is the 15th allocation that was not explicitly freed when the program terminated.

Malloc returned the address 0x3060DFF8 to the application and it was allocated from heap 0.

The stack is listed in reverse order, ie. Most recent stack frame first.

NOTE: The output file is *concatenated* to! It is not overwritten. Each time a command exits and adds “Current allocation report:” before dumping it records.

Case Study: Producer/Consumer

- **Application has 2 threads:**
  - Producer creates elements on a queue malloc'ing memory as needed
  - Consumer processes elements from queue and frees memory
- **Symptoms shows heap segment grows over time from 40 pages to several hundred pages**
- **Debug malloc output results in an empty file**
- **Using 'MALLOCOPTIONS=disclaim' results in no memory growth**
- **disclaim directs VMM to dereference freed memory**

Case Study: Producer/Consumer

- **Deeper detailed analysis showed that the producer was getting bursts of CPU that allowed it to outpaced the consumer for very short periods**
- **As the queue grew, memory was malloced and touched causing the growth in the inuse count**
- **Since all the allocated memory was in fact freed on exit, debug malloc showed no unfreed mallocs**
- **Disclaiming pages when they were freed resulting in no growth of memory frames**
- **Short answer – NO MEMORY LEAK here but the application behavior was not understood**

Case Study: Sweeping the floor

- **Over time, a webserver shows growth in memory inuse count**
- **Multiple add-on modules in the webserver**
- **Webserver owners unable to find cause**
- **Debug Malloc generated output – 1.6 GB – and had multiple sets of data**
- **Used technique of forking and then exiting to keep main process alive but generate debug malloc on demand**
- **Developed program to summarize debug malloc data – too much to analyze otherwise**

Case Study: Sweeping the floor – debug malloc

- **Used the full stack trace to identify complete unique location where memory was being malloc'ed**
- **Totaled instances the stack was seen and number of bytes allocated**
- **Used gensyms output to get symbols in the kernel based on stack addresses**
- **Targeted investigation based on number of bytes allocated and number of times a stack was seen**

Case Study: Sweeping the floor - gensyms

- **gensysms dumps the symbols and addresses from kernel and kernel extentions**
- **Output looks like:**

```
Src: ../../../../../src/bos/kernel/proc/proc_ras.c
00051d60 T .proc_ras_register 60
00051dc0 T .sysproc_init_early 220
00051fe0 T .proc_ras_register_early 100
000520e0 T .proc_ras_init 40
00052120 T .proc_ras_callback 40
00052160 T .gen_proc_ras_callback 140
```

- **Format is ‘address symbol_type symbol_name’**
- **Source code file is also listed**

Case Study: Sweeping the floor - gensyms

- **Use address found in the debug malloc output to resolve symbol in kernel/kernel extention**
- **Address 0x00051fe9 is between 0x00051fe0 and 0x000520e0**
- **Code is put in memory in increasing order:**

  000520e0 T .proc_ras_init 40

  00052120 T .proc_ras_callback 40

  proc_ras_init starts at 0x520e0 and ends at 0x5211f
- **So 0x00051fe9 is in the .proc_ras_register_early routine:**

  00051fe0 T .proc_ras_register_early 100

  000520e0 T .proc_ras_init 40
- **Does not get application addresses/symbols**

Case Study: Sweeping the floor – findings

- **Debug malloc data showed that _pthread_body was being called an increasing number of times without freeing memory**
- **_pthread_body is the starting point of all threads**
- **The number of threads in the ps output did not vary by much**
- **This means new threads were being created but after they terminated the thread memory was not being cleaned up**

Case Study: Sweeping the floor - Conclusion

- **Threads need to be joined or detached to correctly cleanup**
- **Since the memory was left allocated, we checked with the various developers for the modules and found where 1 module was not aware of the join/detach requirement**
- **After the fix was implemented, the memory usage quit growing**

Case Study – When to re-use

- **Application shows memory growth that does not stop until all memory is exhausted**
- **Application is customer developed and critical to their business**
- **svmon shows memory segment growth only in application segments**
- **Unable to use debug malloc to generate output file because the application prevents it and will not start**

Case Study – When to re-use – dbx and malloc

- **Utilize dbx's ability to dump debug malloc records**
- **Start app, attach to it with dbx, dump data and **detach** (exit will terminate the app)**
- **Extended data can be made available:**

  export MALLOCDEBUG=log:extended,stack_depth:32
- **Use dbx 'malloc' command to see summary**
- **Use dbx 'malloc allocation' command to show detailed records**

Case Study – When to re-use – dbx malloc

(dbx) malloc
The following options are enabled:

Implementation Algorithm..... Default Allocator (Yorktown)
Malloc Catch Overflow
Alignment..... 8
Malloc Log
Stack Depth..... 32
Extended Log Data

Statistical Report on the Malloc Subsystem:
Heap 0
heap lock held by..... UNLOCKED
bytes acquired from sbrk()..... 1048592
bytes in the freespace tree..... 1040336
bytes held by the user..... 8256
allocations currently active..... 1
allocations since process start.. 1

The Process Heap
Initial process brk value..... 0x20067bd0
current process brk value..... 0x20167be0
sbrk()s called by malloc..... 1

Case Study – When to re-use – dbx malloc

(dbx) malloc allocations

```
Allocations Held by the Process:

           ADDRESS         SIZE HEAP      PID  PTHREAD_T  CLOCKTIME      SEQ STACK TRACEBACK
0x00000001100cdb30         1232    0   241878 0x00000000 1244387402        0 0x00900000000190e88 malloc_common_...
                                                                             0x00900000000060664 init_malloc
                                                                             0x00900000000061794 malloc
                                                                             0x00900000000715824 __pth_init
                                                                             0x00000000100000780 call_pth_init
                                                                             0x00000000100000694 __threads_init
                                                                             0x00000000100000230 __start
```

- **Note new fields:**
  - HEAP – heap number the memory was allocated from
  - PTHREAD_T – thread ID within process
  - CLOCKTIME – time the malloc was called

Case Study – When to re-use – findings

- **New connections were being created for each connection based on the stacks**
- **Connections were never being reused**
- **Analysis of the application and connection details showed that the application started looking at connection 0 to find an open connection to use**
- **If connection 0 was not available, a new connection was created**
- **Connection 0 was being used as a persistent connection by the application**
- **Since 0 was always in use, all other connections were created as new connections even though connections 1 and higher were unused**

Case study 4 – Failed to maintain

- **Every 2 to 3 months, the system shows a sudden memory growth.**
- **The issue cannot be tracked down to any single application or job.**
- **‘svmon’ data is collected at the beginning and end of every job scheduled. The growth is proven to be sudden but also occurs on many different jobs.**
- **Dumps are taken to review the data in the segments that are growing. No real details are found here.**

Case study 4 – Failed to maintain

- **Memory can only be recovered by rebooting the system.**
- **The svmon data shows the growth to be in mbuf data. Mbuf data is used by the networking system for IO. It is pinned by default.**
- **The final solution was found by extensive digging in a kernel dump.**

Case Study 4 – Failed to maintain

- **NETM NOT AWOKEN SO MEMORY NEVER RETURNED TO SYSTEM**
- **APARs are:**

IZ75391 U834150 shipped 6100-03-05-1015 bos.mp64 6.1.3.5
IZ75409 U834072 shipped 6100-02-08-1015 bos.mp64 6.1.2.8
IZ75459 U831391 shipped 6100-01-09-1015 bos.mp64 6.1.1.9
IZ71092 U830243 shipped 5300-09-07-1015 bos.mp64 5.3.9.7
IZ71092 U830244 shipped 5300-09-07-1015 bos.mp 5.3.9.7
IZ73835 U832236 shipped 5300-08-10-1015 bos.mp 5.3.8.10
IZ73835 U832237 shipped 5300-08-10-1015 bos.mp64 5.3.8.10
IZ73587 U830275 shipped 5300-12 bos.mp 5.3.12.1
IZ73587 U830276 shipped 5300-12 bos.mp64 5.3.12.1
IZ73636 U832822 shipped 5300-11-04-1015 bos.mp 5.3.11.4
IZ73636 U832823 shipped 5300-11-04-1015 bos.mp64 5.3.11.4
IZ73725 U830147 shipped 5300-10-04-1015 bos.mp64 5.3.10.4
IZ73725 U830148 shipped 5300-10-04-1015 bos.mp 5.3.10.4

Total time to resolve was over 1 year due to the problem happening only every 3 months and it's unpredictability.

Conclusions

- **Memory growth can be caused by many different things**
- **The analysis to reach root cause takes significant time, skills and knowledge of the application**
- **Debug malloc can be extremely useful but it also generates a large amount of data to be reduced**
- **There are other tools that can help analyze memory growth**
- **Rarely is the growth of memory due to a problem in AIX – it does happen occasionally**

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Not all common law marks used by IBM are listed on this page. Failure of a mark to appear does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.

Those trademarks followed by ® are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States.

For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

*, AS/400®, e business(logo)®, DBE, ESCO, eServer, FICON, IBM®, IBM (logo)®, iSeries®, MVS, OS/390®, pSeries®, RS/6000®, S/30, VM/ESA®, VSE/ESA, WebSphere®, xSeries®, z/OS®, zSeries®, z/VM®, System i, System i5, System p, System p5, System x, System z, System z9®, BladeCenter®

The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.