NTIC : Using a customized malloc on AIX 5.2 and 5.3

This page last changed on Jun 24, 2008 by sable.

Sungard GP3 uses a virtual machine called "Magnum Runtime" to execute applications written with an in-house language called ADL and compiled to some bytecode representation. Our application can be used by hundreds of clients at the same time for many hours a day on various architectures including AIX 5.2 and 5.3. As a result the memory consummation of our virtual machine is a critical parameter that can have a huge impact on the memory requirements of the server.

Lately we have worked on some optimization of our virtual machine; those optimizations require a analysis of the ADL bytecode which consumes an important amount of mermory at launch time, but which brings big performances improvements later. This important allocation of memory shoud not be a problem as it only happens at launch time for a short lap of time, and it gets freed after that. Unfortunately, we have observed on AIX systems that once our virtual machine has allocated some memory for the optimization phase, that memory never gets freed afterwhile. This behavior is not observed on other systems like Linux.

We first supposed this was a memory leak, but after quite some time of analysis, we have actually come to the conclusion that this is a system problem as can be illustrated by the test cases that we provide below: AIX never actually releases the memory that is freed by our application.

Later we found about the MALLOCOPTIONS=disclaim environment variable which actually forced the system to release the memory. While this allowed our application to correctly release memory, this came with a high cost: the performances when running with this option were extremely deteriorated as can be shown by another test case below.

We have also come with a way to bypass this behavior by using a customized malloc implementation. This malloc is based on some work by Doug Lea as described in his document A Memory Allocator. The malloc implementation described in this document can be downloaded from: ftp://g.oswego.edu/pub/misc/malloc.c

When using this customized malloc, we have found that the memory actually got freed when it should be, as is illustrated by the test cases below.

Illustration of the problem

Initial test to show that free does not actually make memory available to the system.

The test application is a trivial C test case which runs on AIX 5.3. This application will allocate then free a memory area:

```
sable@sirius test$ cat alloc1.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  char *ptr = NULL;
  int size = 128 * 1024 * 1024;
  int i;
  int num;

  printf("initial state\n");
  scanf("%d", &num);
  ptr = malloc(size);
  for(i = 0; i < size; i++)
    *(ptr + i) = 1;
  printf("allocated %d bytes\n", size);
  scanf("%d", &num);
  free(ptr);
  printf("after free\n");
  scanf("%d", &num);
  return 0;
}
```

```
sable@sirius test$ cc alloc1.c -o alloc1
sable@sirius test$ ./alloc1
initial state

[2]+  Stopped                 ./alloc1
sable@sirius test$ ps -u sable | grep alloc1
     1187  762020 pts/25  0:00 alloc1
sable@sirius test$ sudo /usr/bin/svmon -P 762020

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
  762020 alloc1           23525     7148    19804    65252      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB       7429       7148       3836      11220
     m  64 KB       1006          0        998       3377

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   6f0ad         d work shared library text          m   1006     0  998  3377
       0         0 work kernel                       s   7396  7145 3836 11189
    7966         2 work process private              s     21     3    0    21
   ed43b         f work shared library data          s     10     0    0    10
   e65da         1 pers code,/dev/hd1:49163          s      2     0    -     -
sable@sirius test$ fg
./alloc1
1
allocated 134217728 bytes

[2]+  Stopped                 ./alloc1
```

svmon correctly shows that the memory has been allocated

```
sable@sirius test$ sudo /usr/bin/svmon -P 762020

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
  762020 alloc1           56252     7148    19804    98027      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB      40204       7148       3836      43995
     m  64 KB       1003          0        998       3377

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
    7966         2 work process private              s  32789     3    0 32789
   6f0ad         d work shared library text          m   1003     0  998  3377
       0         0 work kernel                       s   7396  7145 3836 11189
   ed43b         f work shared library data          s     17     0    0    17
   e65da         1 pers code,/dev/hd1:49163          s      2     0    -     -
```

However when the memory is freed by the application, the system never actually releases it:

```
sable@sirius test$ fg
./alloc1
1
after free

[2]+  Stopped                 ./alloc1
sable@sirius test$ sudo /usr/bin/svmon -P 762020

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
  762020 alloc1           56188     7148    19804    98027      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB      40204       7148       3836      43995
     m  64 KB        999          0        998       3377
```

```
    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
    7966         2 work process private              s  32789     3    0 32789
   6f0ad         d work shared library text          m    999     0  998  3377
       0         0 work kernel                       s   7396  7145 3836 11189
   ed43b         f work shared library data          s     17     0    0    17
   e65da         1 pers code,/dev/hd1:49163          s      2     0    -     -
```

Even after waiting a long time, on a busy system, the memory never actually gets released. Some additional tests, with the help of an IBM expert showed that at best the memory can be put in swap when the system memory is really busy, and that this memory in swap may be retrieved by the system with a huge cost on performances.

```
sable@sirius test$ sleep 1000; sudo /usr/bin/svmon -P 762020

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
  762020 alloc1           56188     7148    19804    98027      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB      40204       7148       3836      43995
     m  64 KB        999          0        998       3377

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
    7966         2 work process private              s  32789     3    0 32789
   6f0ad         d work shared library text          m    999     0  998  3377
       0         0 work kernel                       s   7396  7145 3836 11189
   ed43b         f work shared library data          s     17     0    0    17
   e65da         1 pers code,/dev/hd1:49163          s      2     0    -     -
```

Using disclaim to release memory

The system environment variable MALLOCOPTIONS=disclaim can be used to force the system to release the memory when the application does a free.

```
sable@sirius test$ MALLOCOPTIONS=disclaim ./alloc1
initial state

1
allocated 134217728 bytes

[3]+  Stopped                 MALLOCOPTIONS=disclaim ./alloc1
sable@sirius test$ ps -u sable | grep alloc1
     1187 1528026 pts/25  0:01 alloc1
sable@sirius test$ sudo /usr/bin/svmon -P 1528026

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
 1528026 alloc1           56188     7148    19804    98027      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB      40204       7148       3836      43995
     m  64 KB        999          0        998       3377

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   7c78c         2 work process private              s  32789     3    0 32789
   6f0ad         d work shared library text          m    999     0  998  3377
       0         0 work kernel                       s   7396  7145 3836 11189
   fdab9         f work shared library data          s     17     0    0    17
   e65da         1 pers code,/dev/hd1:49163          s      2     0    -     -
sable@sirius test$ fg
MALLOCOPTIONS=disclaim ./alloc1
1
after free

[3]+  Stopped                 MALLOCOPTIONS=disclaim ./alloc1
```

```
sable@sirius test$ sudo /usr/bin/svmon -P 1528026

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
 1528026 alloc1           23420     7148    19804    65259      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB       7436       7148       3836      11227
     m  64 KB        999          0        998       3377

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   6f0ad         d work shared library text          m    999     0  998  3377
       0         0 work kernel                       s   7396  7145 3836 11189
   7c78c         2 work process private              s     21     3    0    21
   fdab9         f work shared library data          s     17     0    0    17
   e65da         1 pers code,/dev/hd1:49163          s      2     0    -     -
```

This is the behavior that would be expected by default when allocating and freeing some memory in an application.

This come however with a huge cost: we assist to a huge deterioration of performances, as is illustrated by the following test case:

```
sable@sirius test$ cat alloc2.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  char *ptr = NULL;
  int size = 1024 * 1024;
  int i, j;
  int num;

  for(j = 0; j < 1024 * 1024; j++) {
      ptr = malloc(size);
      free(ptr);
  }
  printf("finished loop %d\n", j);
  return 0;
}
sable@sirius test$ cc -o alloc2 alloc2.c
sable@sirius test$ /usr/bin/time ./alloc2
finished loop 1048576

Real   0.40
User   0.16
System 0.00
sable@sirius test$ MALLOCOPTIONS=disclaim /usr/bin/time ./alloc2
finished loop 1048576

Real   121.56
User   4.77
System 44.87
```

With the MALLOCOPTION=disclaim option, this simple test case will take more than 300% more time to complete. We did some tests with real applications, and we observed the same deterioration of performances. So even if this option ensure that the memory is correctly released, it is not usable on a production server as it generates a huge deterioration of performances.

Customized malloc

This third test uses the exact same test case as in the first example above, but this time it is linked with dlmalloc, the customized malloc.

The file malloc-2.7.2.c can be downloaded from ftp://g.oswego.edu/pub/misc/malloc.c.

dlmalloc uses both sbrk and mmap. The sbrk system call will change the size of the heap to be larger or smaller as needed, while the mmap system call will be used when extremely large segments are allocated. The heap method suffers the same flaws as any other, while the mmap method may avert problems with huge buffers trapping a small allocation at the end after their expiration.

The mmap method has its own flaws: it always allocates a segment by mapping entire pages. Mapping even a single byte will use an entire page
which is usually 4096 bytes. Although this is usually quite acceptable, many architectures provide large page support (4 MiB or 2 MiB with PAE
on IA-32). The combination of this method with large pages can potentially waste vast amounts of memory. The advantage to the mmap
method is that when the segment is freed, the memory is returned to the system immediately.

As we will see, the memory is correctly released to the system.

We link the source with dlmalloc:

```
sable@sirius test$ cc alloc1.c malloc-2.7.2.c -o alloc3
alloc1.c:
malloc-2.7.2.c:
sable@sirius test$ ./alloc3
initial state

[1]+  Stopped                 ./alloc3
sable@sirius test$ ps -u sable | grep alloc3
     1187 2654254  pts/6  0:00 alloc3
sable@sirius test$ sudo /usr/bin/svmon -P 2654254

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
 2654254 alloc3           23596     7148    19802    65252      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB       7436       7148       3834      11220
     m  64 KB       1010          0        998       3377

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   6f0ad         d work shared library text          m   1010     0  998  3377
       0         0 work kernel                       s   7399  7145 3834 11189
   575ec         2 work process private              s     21     3    0    21
    6906         f work shared library data          s     10     0    0    10
   b6111         1 pers code,/dev/hd1:49168          s      6     0    -     -
sable@sirius test$ fg
./alloc3
1
allocated 134217728 bytes

[1]+  Stopped                 ./alloc3
sable@sirius test$ sudo /usr/bin/svmon -P 2654254

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
 2654254 alloc3           56372     7148    19802    98028      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB      40212       7148       3834      43996
     m  64 KB       1010          0        998       3377

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   a4db7         - work mmap paging                  s  32769     0    0 32769
   6f0ad         d work shared library text          m   1010     0  998  3377
       0         0 work kernel                       s   7399  7145 3834 11189
   575ec         2 work process private              s     21     3    0    21
    6906         f work shared library data          s     17     0    0    17
   b6111         1 pers code,/dev/hd1:49168          s      6     0    -     -
    d082         3 mmap maps 1 source(s)             s      0     0    -     -
```

The memory is allocated and accessed just like in the previous test case.

```
sable@sirius test$ fg
./alloc3
1
after free

[1]+  Stopped                 ./alloc3
sable@sirius test$ sudo /usr/bin/svmon -P 2654254

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
 2654254 alloc3           23795     7148    19802    65259      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB       7443       7148       3834      11227
     m  64 KB       1022          0        998       3377

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   6f0ad         d work shared library text          m   1022     0  998  3377
       0         0 work kernel                       s   7399  7145 3834 11189
   575ec         2 work process private              s     21     3    0    21
    6906         f work shared library data          s     17     0    0    17
   b6111         1 pers code,/dev/hd1:49168          s      6     0    -     -
```

However this time, when free is called in the application, the memory is directly released and available to the system:

Concerning performances, they are a least 2 times better than disclaim on our simple test case:

```
sable@sirius test$ cc alloc2.c malloc-2.7.2.c -o alloc4
alloc2.c:
malloc-2.7.2.c:
sable@sirius test$ /usr/bin/time ./alloc4
finished loop 1048576

Real   56.96
User   8.12
System 13.32
```

Some tests on real applications allocating various size of buffers have shown that the performances are actually quite good, while MALLOCOPTIONS=disclaim performances are very bad.

Conlusion

In the context of our application which can have many hundred instances running for hours, this difference in behavior can dramatically reduce the memory requirements on the server and increase performances.

We would appreciate to hear from IBM experts if this problem has already been observed for some other applications, and how it has been handled until now? Also we would like to know what is your position concerning the use of an customized malloc in applications running on AIX.