## NTIC : Using a customized malloc on Solaris 8 and 10.

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 Solaris 8 and 10. 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 memory at launch time, but which brings big performances improvements later. This important allocation of memory should 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 Solaris 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: Solaris never actually releases the memory that is freed by our application.

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 the 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 Solaris 10. This application will allocate then free a memory area:

```
neptune:~/ssa/test$ cat alloc1.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  char *ptr = NULL;
  int size = 1024 * 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;
}
neptune:~/ssa/test$ cc alloc1.c -o alloc1
neptune:~/ssa/test$ ./alloc1
initial state
^Z
```

```
[1]+  Stopped                 ./alloc1
neptune:~/ssa/test$ ps -u recette | grep alloc1
 25294 pts/37      0:00 alloc1
neptune:~/ssa/test$ pmap -x 25294
25294:  ./alloc1
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
00010000       8       8       -       - r-x--  alloc1
00020000       8       8       8       - rwx--  alloc1
FF280000     864     808       -       - r-x--  libc.so.1
FF368000      32      32      32       - rwx--  libc.so.1
FF370000       8       8       8       - rwx--  libc.so.1
FF380000       8       8       8       - rwx--    [ anon ]
FF390000       8       8       -       - r-x--  libc_psr.so.1
FF3A0000      24      16      16       - rwx--    [ anon ]
FF3B0000     184     184       -       - r-x--  ld.so.1
FF3EE000       8       8       8       - rwx--  ld.so.1
FF3F0000       8       8       8       - rwx--  ld.so.1
FFBFC000      16      16      16       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb    1176    1112     104       -
neptune:~/ssa/test$ fg
./alloc1
1
allocated 1073741824 bytes
^Z
[1]+  Stopped                 ./alloc1
```

pmap correctly shows that the memory has been allocated

```
neptune:~/ssa/test$ pmap -x 25294
25294:  ./alloc1
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
00010000       8       8       -       - r-x--  alloc1
00020000       8       8       8       - rwx--  alloc1
00022000   32632   32632   25808       - rwx--    [ heap ]
02000000 1048576 1048576 1048576       - rwx--    [ heap ]
FF280000     864     800       -       - r-x--  libc.so.1
FF368000      32      32      32       - rwx--  libc.so.1
FF370000       8       8       8       - rwx--  libc.so.1
FF380000       8       8       8       - rwx--    [ anon ]
FF390000       8       8       -       - r-x--  libc_psr.so.1
FF3A0000      24      16      16       - rwx--    [ anon ]
FF3B0000     184     184       -       - r-x--  ld.so.1
FF3EE000       8       8       8       - rwx--  ld.so.1
FF3F0000       8       8       8       - rwx--  ld.so.1
FFBFC000      16      16      16       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb 1082384 1082312 1074488       -
```

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

```
neptune:~/ssa/test$ fg
./alloc1
1
after free
^Z
[1]+  Stopped                 ./alloc1
neptune:~/ssa/test$ pmap -x 25294
25294:  ./alloc1
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
00010000       8       8       -       - r-x--  alloc1
00020000       8       8       8       - rwx--  alloc1
00022000   32632   32632   25808       - rwx--    [ heap ]
02000000 1048576 1048576 1048576       - rwx--    [ heap ]
FF280000     864     808       -       - r-x--  libc.so.1
FF368000      32      32      32       - rwx--  libc.so.1
FF370000       8       8       8       - rwx--  libc.so.1
```

```
FF380000       8       8       8       - rwx--    [ anon ]
FF390000       8       8       -       - r-x--  libc_psr.so.1
FF3A0000      24      16      16       - rwx--    [ anon ]
FF3B0000     184     184       -       - r-x--  ld.so.1
FF3EE000       8       8       8       - rwx--  ld.so.1
FF3F0000       8       8       8       - rwx--  ld.so.1
FFBFC000      16      16      16       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb 1082384 1082320 1074488       -
```

Even after waiting a long time, on a busy system, the memory never actually gets released. Some additional tests, with the help of a Sun 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.

```
neptune:~/ssa/test$ sleep 1000; pmap -x 25294
25294:  ./alloc1
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
00010000       8       8       -       - r-x--  alloc1
00020000       8       8       8       - rwx--  alloc1
00022000   32632   32632   25808       - rwx--    [ heap ]
02000000 1048576 1048576 1048576       - rwx--    [ heap ]
FF280000     864     808       -       - r-x--  libc.so.1
FF368000      32      32      32       - rwx--  libc.so.1
FF370000       8       8       8       - rwx--  libc.so.1
FF380000       8       8       8       - rwx--    [ anon ]
FF390000       8       8       -       - r-x--  libc_psr.so.1
FF3A0000      24      16      16       - rwx--    [ anon ]
FF3B0000     184     184       -       - r-x--  ld.so.1
FF3EE000       8       8       8       - rwx--  ld.so.1
FF3F0000       8       8       8       - rwx--  ld.so.1
FFBFC000      16      16      16       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb 1082384 1082320 1074488       -
```

# Customized malloc

This second test uses the exact same test case as in the 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](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:

```
neptune:~/ssa/test$ cc alloc1.c malloc-2.7.2.c -o alloc2
alloc1.c:
malloc-2.7.2.c:
neptune:~/ssa/test$ ./alloc2
initial state
^Z
[2]+  Stopped                 ./alloc2
```

```
neptune:~/ssa/test$ ps -u recette | grep alloc2
 29207 pts/37      0:00 alloc2
neptune:~/ssa/test$ pmap -x 29207
29207:  ./alloc2
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
00010000      16      16       -       - r-x--  alloc2
00022000       8       8       8       - rwx--  alloc2
FF280000     864     808       -       - r-x--  libc.so.1
FF368000      32      32      32       - rwx--  libc.so.1
FF370000       8       8       8       - rwx--  libc.so.1
FF380000       8       8       -       - r-x--  libc_psr.so.1
FF390000       8       8       8       - rwx--    [ anon ]
FF3A0000      24      16      16       - rwx--    [ anon ]
FF3B0000     184     184       -       - r-x--  ld.so.1
FF3EE000       8       8       8       - rwx--  ld.so.1
FF3F0000       8       8       8       - rwx--  ld.so.1
FFBFC000      16      16      16       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb    1184    1120     104       -
neptune:~/ssa/test$ fg
./alloc2
1
allocated 1073741824 bytes
^Z
[2]+  Stopped                 ./alloc2
neptune:~/ssa/test$ pmap -x 29207
29207:  ./alloc2
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
00010000      16      16       -       - r-x--  alloc2
00022000       8       8       8       - rwx--  alloc2
BF000000   16384   16384   12544       - rw---    [ anon ]
C0000000 1015808 1015808 1015808       - rw---    [ anon ]
FE000000   16392   16392   16344       - rw---    [ anon ]
FF280000     864     800       -       - r-x--  libc.so.1
FF368000      32      32      32       - rwx--  libc.so.1
FF370000       8       8       8       - rwx--  libc.so.1
FF380000       8       8       -       - r-x--  libc_psr.so.1
FF390000       8       8       8       - rwx--    [ anon ]
FF3A0000      24      16      16       - rwx--    [ anon ]
FF3B0000     184     184       -       - r-x--  ld.so.1
FF3EE000       8       8       8       - rwx--  ld.so.1
FF3F0000       8       8       8       - rwx--  ld.so.1
FFBFC000      16      16       8       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb 1049768 1049696 1044792       -
```

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

```
neptune:~/ssa/test$ fg
./alloc2
1
after free
^Z
[2]+  Stopped                 ./alloc2
```

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

```
neptune:~/ssa/test$ pmap -x 29207
29207:  ./alloc2
 Address  Kbytes     RSS    Anon  Locked Mode   Mapped File
00010000      16      16       -       - r-x--  alloc2
00022000       8       8       8       - rwx--  alloc2
FF280000     864     800       -       - r-x--  libc.so.1
FF368000      32      32      32       - rwx--  libc.so.1
FF370000       8       8       8       - rwx--  libc.so.1
```

```
FF380000       8       8       -       - r-x--  libc_psr.so.1
FF390000       8       8       8       - rwx--    [ anon ]
FF3A0000      24      16      16       - rwx--    [ anon ]
FF3B0000     184     184       -       - r-x--  ld.so.1
FF3EE000       8       8       8       - rwx--  ld.so.1
FF3F0000       8       8       8       - rwx--  ld.so.1
FFBFC000      16      16       8       - rw---    [ stack ]
-------- ------- ------- ------- -------
total Kb    1184    1112      96       -
```

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 Sun 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 Solaris.