From: Spitzer, Andras ([redacted])
Sent: Tuesday, April 05, 2011 12:41 AM
To: [redacted] ([redacted])
Cc: [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted]); [redacted] ([redacted])
Subject: RE: GIS - Incident Management - #6637502| gejpfk1| box is up after reboot

Mayumi, World,

Please find the RCA below about this case.

Symptom :

Last Friday we received a case when after a printer installation on gejpfk1 **Oracle DB stopped working**, even after a fresh restart Oracle still wasn't able to start up.

Analysis :

After I had a discussion with World (DBA of the Oracle DBs) he showed me the error messages he saw also he told me that java stopped working as well, and this may either be a coincident but also can be part of the problem. The Oracle error message was very unclear :

```
SQL*Plus: Release 9.2.0.8.0 - Production on Mon Apr 4 19:50:45 2011

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

Connected to an idle instance.

SQL> startup nomount
ORA-27103: internal error
```

I looked up every possible article/metalink doc about this error code, but **found nothing useful**, most likely the "internal error" also indicates that the problem for some reason is not well documented. These articles were mainly about resource problems, and World also sent me a resource related error message so my first thougth was to **focus on the basic resource areas**. I checked the open file descriptors available, soft limit, hard limit, all the shared memory and shared segment parameters, and even though I found a few suspicious values none of them turned out to be the real root cause. I also turned on the resource violation reporting facility in Solaris 10, and I was hoping to see some OS warnings about why the Oracle couldn't start. Nothing came..

Also the java error was telling me that the **problem still might be around resources** :

```
$ java -version
Java HotSpot(TM) Client VM warning: Attempt to allocate stack guard pages failed.
Error occurred during initialization of VM
Could not reserve enough space for code cache
```

Allocation error. This would indicate either memory shortage, or some shared memory parameter limitation. Again, the **resource violation reporting logged nothing**. At this point I felt a bit disappointed, but in the same time I started to think of other areas how to move further. I started to trace both Oracle and the java binaries, to look for anything suspicious. It's pretty difficult to trace Oracle because even during standard operation it reads/writes many error messages internally, so it's not easy to pick the ones that are actually can be relevant in our case. After analyzing both trace outputs, from Oracle and java, I **found something which was weird** :

```
...
15455/1:        getrlimit(RLIMIT_NOFILE, 0xFFBFC458)            = 0
15455/1:                cur = 16384  max = 16384
15455/1:        setrlimit(RLIMIT_NOFILE, 0xFFBFC458)            = 0
15455/1:                cur = 16384  max = 16384
15455/1:        open("/dev/zero", O_RDONLY)                     = 9
15455/1:        mmap(0x00000000, 524288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_NORESERVE, 9, 0) Err#11 EAGAIN
15455/1:        close(9)                                        = 0
```

```
15455/1:        shmdt(0x20000000)                               = 0
15455/1:        shmget(1759181964, 0, 0)                        = 43
15455/1:        shmctl(43, IPC_STAT, 0xFFBFC050)                = 0
15455/1:                u=988   g=961   cu=988   cg=961   m=0100640 seq=0
key=1759181964
...
```

An **mmap(2) system call, which returns with a EAGAIN** error.. That doesn't look good at all, not sure if that's a problem though. mmap is responsible for mapping files to the memory area, it can be used in many ways, but usually it's used to map dynamic libraries to memory hence providing a memory cache for the frequently used dynamic library files. (File caching is a central piece of Solaris' Virtual Memory management) That's not the only use of course, in this case (in Oracle) **mmap is using /dev/zero to create a 524288byte size memory area** filled with zeros (because /dev/zero provides 0s). This is a generic behaviour so this should not get an EAGAIN. The interesting part is that I was able to discover a similar pattern in the java's trace output :

```
...
28615/1:        close(5)                                        = 0
28615/1:        brk(0x00089208)                                 = 0
28615/1:        brk(0x0008D208)                                 = 0
28615/1:        stat("/usr/j2se/jre/classes", 0xFFBFEC00)       Err#2 ENOENT
28615/1:        mmap(0x40000000, 33554432, PROT_NONE, MAP_PRIVATE|MAP_NORESERVE, 3, 0)
Err#11 EAGAIN
28615/1:        write(1, 0xFFBFE4C0, 43)                        = 43
28615/1:           E r r o r   o c c u r r e d   d u r i n g   i n i t i a l i z a
28615/1:           t i o n   o f   V M\n
28615/1:        write(1, 0xFEFDC169, 45)                        = 45
28615/1:           C o u l d   n o t   r e s e r v e   e n o u g h   s p a c e   f
...
```

At this point I started to be suspicious about mmap(2), but because I saw mmap(2) works hundreds of times even in these trace outputs, I had to find out **why these mmaps are different**, what is common between these two mmaps, that breaks both programs. As I already had a hunch, I started to look up in the java's trace what file is mapped here into the memory. I only had to scroll back to see which open() will receive the file descriptor 3. (The 5th parameter for mmap(2) is the file descriptor which points to the file you want to map into memory) The result was :

```
...
28615/1:        open("/dev/zero", O_RDWR)                       = 3
...
```

Not even surprised, maybe a bit happy.. Ok, so it looks like that **whenever mmap is called with /dev/zero, it fails**. That sounds bad, as it's a very common practice, this can explain why Oracle and java can't really start, also explains why Oracle doesn't have a detailed error message about it. Because it's really unexpected, it rarely happens. When does this happen? Let me paste here the relevant parts from mmap(2)'s man page :

```
...

     EAGAIN          The mapping could not be locked in memory.

                     There was insufficient room to reserve  swap
                     space for the mapping.
...
```

Ok. So this goes back to my theory of having some sort of resource shortage. **Still, I found none**. This error should pop up whenever the system runs out of swap space. We had plenty still, actually disk swap was not even used, not a single block! What is going on here? Before I moved on, I decided that I want to test this if indeed that's the line causing us problem. So, I wrote a 3 liner C program, which did nothing but opened /dev/zero, mmap'd it, and closed the file descriptor. Compiled it with GCC, ran it on fk1 and fk2, also on a few other lab servers. I found out **that's exactly the line which behaves differently on gejpfk1**! At this point I was really happy because I **found a reproducible part of the issue I can focus on**, also I felt I may have problems moving on from here.

So, now **I was sure that the problem comes when mmap(2) is trying to map /dev/zero**. Luckily a lot of question came to my mind, like is it /dev/zero which causing this problem or the mmap(2) itself? Of course the real question was why is this happening, but we have to take one step at a time. The dilemma to me was how can I debug further an issue which is at this point shortlisted to a single system call? Also, because this system call is part of the libc, the source code is not even available on the OpenSolaris source repository. My only solution left is our old friend, DTrace. I created a short **DTrace script which showed me all the probes in my little test program**, and DTrace's is way more detailed

than the system calls.

Please see **what happens with mmap when it works**, I ran this on a different system (on gejpfk2), which I knew mmap is working properly :

```
 28  -> smmap32
 28    -> getf
 28      -> set_active_fd
 28      <- set_active_fd
 28    <- getf
 28    -> smmap_common
 28      -> fop_map
 28        -> spec_map
 28          -> vn_has_flocks
 28          <- vn_has_flocks
 28          -> spec_char_map
 28            -> cdev_segmap
 28              -> mmsegmap
 28                -> getminor
...
```

See what happens with **mmap on the troubled system**, gejpfk1 :

```
 28  -> smmap32
 28    -> getf
 28      -> set_active_fd
 28      <- set_active_fd
 28    <- getf
 28    -> smmap_common
 28      -> fop_map
 28        -> spec_map
 28          -> vn_has_flocks
 28          <- vn_has_flocks
 28        <- spec_map
 28      <- fop_map
 28    <- smmap_common
...
```

Now, most of these probes can be found in the OpenSolaris source repository, so I had the chance to read these functions, but it was almost unnecessary. The names speaks for themselves. The difference comes after "vn_has_flocks" function (you see the arrow points to right on the correect one indicating that the program path continued below, and left on the failing one indicating that the program flow didn't continue, but returned), which **indicates that the file has locks** on it, **that's why mmap won't even continue mapping /dev/zero on gejpfk1**. That's getting interesting, file locks on /dev/zero.. After that it getting easier, I looked up the processes keeping ./dev/zero open :

```
gejpfk1% /usr/sbin/fuser /dev/zero
/dev/zero:      645o     644o     643o     601o     599o
```

Further **resolving these process IDs** :

```
gejpfk1% ps -ef | grep 599
    root   599   521   0 16:35:54 ?           0:00 /usr/lib/lp/local/lpsched
    root   601   599   0 16:35:54 ?           0:00 sh /etc/lp/interfaces/gejpfk1
gejpfk1-5 root /etc/hosts 1 dest=gejpfk1:gejpfk1
```

lpsched? The printer daemon? At this point I felt we are really close, but I had to confirm, so I used pfiles to see the open file descriptors of lpsched :

```
gejpfk1# pfiles 643
643:    /usr/spool/lp/bin/lp.tell -l gejpfk1
  Current rlimit: 4096 file descriptors
   0: S_IFIFO mode:0000 dev:337,0 ino:319 uid:0 gid:0 size:0
      O_RDWR
   1: S_IFCHR mode:0666 dev:339,0 ino:6815772 uid:0 gid:3 rdev:13,12
      O_RDWR|O_APPEND
```

```
      advisory write lock set by process 599
      /devices/pseudo/mm@0:zero
   2: S_IFCHR mode:0666 dev:339,0 ino:6815752 uid:0 gid:3 rdev:13,2
      O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE
      /devices/pseudo/mm@0:null
   3: S_IFCHR mode:0666 dev:339,0 ino:6815772 uid:0 gid:3 rdev:13,12
      O_RDWR|O_APPEND
      advisory write lock set by process 599
      /devices/pseudo/mm@0:zero
   5: S_IFCHR mode:0666 dev:339,0 ino:6815752 uid:0 gid:3 rdev:13,2
      O_WRONLY
      /devices/pseudo/mm@0:null
   9: S_IFIFO mode:0000 dev:337,0 ino:302 uid:0 gid:0 size:0
      O_RDWR
```

One small technical note, /dev/null is equal to /devices/pseudo/mm@0:zero (it's a symlink), and as you see file descriptor #1 : " **advisory write lock set by process 599**"

We got it.. ! **lpsched is keeping a write lock on /dev/zero, which is very abnormal**.. Why would it? My senses told me that as lpsched should not lock /dev/zero during standard operation, this may come from a misconfiguration :

```
gejpfk1# cd /etc/lp
gejpfk1# ls -la
total 48
gejpfk1# find . -type f -exec grep -l "/dev/zero" {} \;
./printers/gejpfk1/configuration
```

Got it, let me see :

```
ejpfk1# more ./printers/gejpfk1/configuration
Banner: on
Content types: UXPS
Device: /dev/zero
Interface: /usr/lib/lp/model/ApeosPort4C5570
Printer type: ApeosPort4C5570
Modules:
Options: dest=gejpfk1:gejpfk1,protocol=bsd
```

There you go ! **Device: /dev/zero** ! Now, you should ask me if this is normal. Not really. The trick is that lpsched locks the device if there are more than one printing occurs to the same device (that sounds really obvious), except for network printers. When **you define network printers, you should put "/dev/null" as Device, not "/dev/zero".** The reason is simple, **lpsched has a built in protection against locking /dev/null**, it has no protection against locking /dev/zero... !

I shut down lpsched to **test if my theory works**, and after the printer daemon shutdown, I tried java :

```
$ java -version
java version "1.5.0_18"
Java(TM) Platform, Standard Edition for Business (build 1.5.0_18-b02)
Java HotSpot(TM) Server VM (build 1.5.0_18-b02, mixed mode)
```

Looks good, isn't it ? And I **tried to start Oracle** :

```
$ ./sqlplus "/as sysdba"

SQL*Plus: Release 9.2.0.8.0 - Production on Tue Apr 5 03:49:39 2011

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

Connected to an idle instance.

SQL> startup nomount
ORACLE instance started.

Total System Global Area  598299504 bytes
```

```
Fixed Size                   455536 bytes
Variable Size             419430400 bytes
Database Buffers          167772160 bytes
Redo Buffers               10641408 bytes
```

Lovely! It works ! After that I simply amended the configuration file by replacing /dev/zero with /dev/null, restarted the lpsched daemon, and since then everything looks good! World, please start the database too because after the test 'startup nomount' I shut down the Oracle DB again, so you can start it however you like.

Solution :

Stop the lpsched daemon :

```
# svcadm disable svc:/application/print/server:default
```

Create a backup copy of /etc/lp/printers/gejpfk1/configuration, and **replace /dev/zero with /dev/null**

```
# cd /etc/lp/printers
# cp configuration configuration.20110404
# <vi configuration>
```

Restart the printer daemon :

```
# svcadm enable svc:/application/print/server:default
```

Double check if **1.** printer daemon runs with ps **2.** make sure no process is keeping a lock on /dev/zero with fuser and pfiles (both done and good to go!)

Summary :

Oracle and java couldn't run succesfully because **both were using** the standard practice of **mmap'ing /dev/zero** to a certain memory area so it can be used as a buffer. The problem was that **mmap couldn't run succesfully because /dev/zero had file locks** by lpsched (the printer daemon), which was the consequence of a misconfiguration. One of the printer definitions had /dev/zero given as Device name, instead of /dev/null which is the correct name in case you are configuring network printers. lpsched by default doesn't lock device files, only if there are more than one, simultanous printing on the same device. Because of the misconfiguration lpsched thought that /dev/zero is indeed a local printer device, so it treated like one, kept it locked whenever it was necessary. Because of this design lpsched has a built in protection to never put locks on /dev/null, that's why the correct method is to use /dev/null as a Device in case of network printer configuration.

Regards,
sendai