Subject: Using threads on OpenVZ, and memory allocation versus memory usage
Posted by MvdS on Sun, 06 May 2007 14:07:39 GMT
View Forum Message <> Reply to Message

I noticed NPTL allocates huge amounts of memory for each thread. 8MiB per thread is not uncommon. Of course with a privvmpages limit set on a VE, starting a program with threads can easily consume all privvmpages.

Setting privvmpages is a common (and advised) way to limit the amount of RAM a VE is able to use. But it is both unrelated to the RAM usage and unfair against applications allocating without using the allocated memory, NPTL is a widly used example of this. Furthermore, with a vanilla linux kernel, the allocation is only limited by the address space op the application. This means, on a 32bit system, the typical limit is 3GiB/process. On a typical OpenVZ system this is only privvmpages for the complete VE.

Portable code should not count on the behavior specific to the kernel, however NPTL is made in collaboration with the linux kernel, and does expect this implementation detail.

OpenVZ breaks this behavior, and thereby greatly reduces the usability of threads within a VE.

I've been discussing this behavior with an OpenVZ system administrator, and there seems no sane way to limit the amount of available RAM.
It is possible to use oomguarpages to guarantee some amount of RAM+swap, but it seems not possible to limit it.

With only oomguarpages configured as a limit for system resources the less than ideal situation that one VE uses all RAM+swap, while other VE's are below their limits, is possible. Because of the nature of swapping, this slows down the complete system. Also, the less active VE's might be totally swapped-out, which means reduced responsiveness, because of 1 misbehaving VE.

This might be resolved by implementing a hard limit on the RAM+swap counter, relating to oomguarpages in the same way as privvmpages relates to vmguarpages. The future releases that might support a settable physpages might help, but is not quite the same.

My direct questions about this matter are:
1. Are there any workarounds to emulate the vanilla kernel behavior without losing control of VE's memory usage?
1. If not, are the OpenVZ developers aware of the 'punishment' users of threaded applications get?
2. What is the opinion across the OpenVZ user base about this?
3. Is there any development in progress to remedy this situation?

PS: Here is a sample application to reproduce the behavior:

```
/* to compile:
 * $ LDFLAGS=-lpthread make dummy-threads
 */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <pthread.h>
#define NUM_THREADS 131072

void *thread_routine(void *data){}

int main(int argc, char **argv){
  pthread_t thread;
  int i;
  int r = 0;

  printf("Trying to create %i threads\n", NUM_THREADS);
  for(i=0;i<NUM_THREADS && !r;i++){
    r = pthread_create(&thread, NULL, thread_routine, NULL);
    if(r) perror("pthread_create");
  }
  printf("A total of %i threads created\n", i);
  pause();
}
```

On 32bit linux this typicaly creates 300+ threads, on 64bit linux this in 3000+, but with OpenVZ this depends on privvmpages and other applications running within the VE.
Also, running the program 2 times on linux does not matter, on OpenVZ the second program is only able to start 1 thread.
