`Memtest`: Finding holes in the VM system WIP

Juan Quintela
Department of Computer Science
Universidade da Coruña
quintela@dc.fi.udc.es

January 11, 2001

Abstract

This paper describes the development of a test suite for the VM subsystem and several of the resulting programs in detail. A proposal for dealing with the shown bottlenecks are made. This suite of programs is called `memtest`. The suite is composed of several programs that generate different kind of IO and memory loads, such as writing big files (mmap*), using a lot of shared memory (ipc*), programs that do a lot of memory allocations/frees. This test suite is not used for benchmarking, it is used to find bottlenecks.

In the presentation we discuss the goals of several tests. The part of the kernel they affect and what are the good ways to handle them.

The tests that form the suite has been contributed by several people. The suite intend to be a place to put tests when anybody find a bottleneck and write a program to show it, then it is easy to make sure that future versions don't have that problem.

1 Introduction

This paper describes the development of a test suite for the *VM subsystem*. This test suite was used to let people get for a single place programs for testing the system for errors, and a place where you can put code that found bugs in previous implementations and then see that we don't have the same problem again.

In the section 2 is described the born of the `memtest` suite. In the next sections, I describe several of the tests, what they do and what problems they find, and what was the solution used to solve them.

2 Previous life

In the beginning, the author was an `happy` PhD. student that was working in his PhD. thesis. The programs related with his thesis stressed a lot the *VM layer* and made my linux machine die hard (die in the sense of *Oops*). He used the standard procedure in this cases: He wrote a nice *Bug report* to the *Linux kernel mailing list* waiting for the *nice kernel hackers* to fix his problem. That was in the 2.2 kernel era. But nothing happened. He continued working in his thesis, thinking that the problems will be solved in 2.4 with the new memory layer/page cache .... Each time that **Linus** get out a new kernel version, he tested the new kernel version, it normally solved some problems and other appeared ..... At the end (end???) of the 2.3 era (i.e. `2.3.99-preX`), he found that his problems has not solved yet. Then he thought that it would be a good idea to *try to help* the `kernel hackers` to fix the problem. At the same time, it happened that **Rik van Riel** came to my University to give a couple of conferences. He was the right person to show the problems that he was having. He show him the problems, and he asked for a small program which reproduced the hangs. `Memtest` was born. After I have some programs written, I got other people who asked me to get more programs inside the suite.

3 The tests

One important thing about almost all the tests in `memtest` is that they are test to check that the `VM` layer behaves well, they are not benchmarks in the speed/space sense. It is good that this programs run well, but the important thing is that they should not run *too bad*. In the future, I will try to also add some speed tests to it, or at least to give some pointers to other benchmarks and the way to use them for searching for several bottlenecks.

4 mmap001 tests

The `mmap` tests are examples based in the works that I was doing in my PhD. `mmap001` is a test that creates a file the size of the physical memory of the machine and then writes it sequentially. In the `2.3.99-preX` kernel series, a machine with 128MB of RAM will stall during as much time as 30 seconds because in the kernel, it waited for starting writing something to disk until there was no more free space. At that point the kernel started to write asynchronously the dirty pages, but **all** the pages was dirty. Then it started to write asynchronously the whole memory of the machine and that took a lot of time to succeed. This test is one of the clearest examples that `memtest` is not a benchmark. This test is supposed that for the system, it should be the same mmap a big file and write to it sequentially than do normal writes to the same size file. This test is not needed to run *very fast*, but a normal user will not expect the whole system to stall for minutes. Once that the biggest stalls have been solved there was still problems that the kernel got loads around 15 while running this test, what is also not expected in a normal system.

5 mmap002

This test is a continuation of the previous test. It mmap a file twice the size of the memory, then it writes sequentially the first half of the file. Then it opens an anonymous mapping and copies all the info from the file to the anonymous mapping. After that it copies again the shared mapping to the other half of the file. This test showed that the kernel at that moment begins to swap when it was too late. It waited for doing any swap, writing to disk, etc until there was no more clean pages, at that point, all the system was doing really bad (read trashing). The problem here, is that the system tried too hard caching pages (and specially dirty pages). Other problem was that all the process were having problems for doing allocations, when there was only a memory hog, and it was *supposedly* easy to detect it. Well, at the end it showed that it was not *so easy* to detect the memory hog, when there was only one, and the problem become really nasty when we had several memory hogs.

This is another test, where all the heuristics of the kernel made it to work worst (i.e. we *never* reused a single page of the file, and we walk sequentially files bigger than physical memory. That made that a system that don't do *any* cache will be faster running that test. But for normal use, we prefer a system that do caching. The problem here is that we want to detect when we are accessing a file only sequentially and then not doing *so* much caching. One user could not wait a full speed write with mmap002, but he will also not wait that the system enter trashing when there is only a process that is doing linear copies of files.

6 ipc001

This test was created by Christoph Roland to test *System V shared memory.* It creates several shared memory segments and try to test that all the operations in the system (writes, attach, dettach, ...).

7 Future Work

There are several things that I have planned to do to make the `memtest` suite more useful:

- Put more documentation for the suite, to make easy for other people to use it and more importantly, to understand the results, while they are wrong or correct.
- Get more tests for the suite. Folks, I am accepting submissions of what are the tests/scripts/knowledge that you use for testing the system.

- Write more documentation.
- Make the tests more modular. The idea is to be able to get an easy way to simulate real loads. I want to make easy to define in an easy way tests like misc001, where there are one process making `malloc()/free()` for 1/3 of the memory, uses `mmap()` for other third of the memory and `fwrite()` for other file. That would be easy if the test where very simple, getting their parameters from the command line.
- Have I told about having documentation to let other people doing the previous kind of thing.
- Include benchmarks (or pointers to it) and document the way to use them for measuring specific things. Just now everybody is using some benchmarks for doing that, but each people use their small subset and there is no way to know what benchmark is used for measuring what, and what is the correct way to configure the benchmarks. One example in this regard is comment things like people use normally `dbench 48` to measure the performance of the file system and the page cache. An idea of what means the parameter, what you should expect and the causes of previous pitfalls will be very useful.
- Considering the possible integration of memtest with **LTP: Linux Test project** (`http://oss.sgi.com/projects/ltp/`). Their is a bigger and more ambitious project, but there is a bit more difficult to write a test for doing that. I am thinking about integrating both efforts (i.e. let them to do all the difficult work), or at least making an easy way to share code.
- Create a way to run the tests *non-interactively.* By non interactively I mean that it should be possible to detect if the system is responsible or not under high load created by the tests. Just now, you have to guess if the system is more/less responsive with a change, and there is not a way to run all the tests at night and know at the following morning if some of the tests made the interactive response bad. Playing music and hearing the pikes don't work when you are out of the office. There is one program that does something similar for the low-latency tests, I could use if for a start.

8 Acknowledgements

I want to thank several people and institutions for the help that they have giving me for doing `memtest`:

- **Rik van Riel**: He explained me a lot of things and helped me a lot to begin working in the Linux Kernel.

- `#kernelnewbies` at `irc.openprejects.net`: There is a lot of cool people connected there that helped me a lot with discussions and explanations about all the questions that I have about *The Linux Kernel*.
- All the people that contributed code and ideas to the suite.
- **Conectiva** and the **Universidade da Coruña** for funding an SMP test machine that helped me to find holes, test bugs and develop the suite (Like in the real life, almost all the bugs shows up really faster when you are working with *SMP*).