OLS 2003 Impressions

Put 500 kernel hackers together, and what do you get?

Muli Ben-Yehuda

`mulix@mulix.org`

IBM Haifa Research Labs

What is OLS

- OLS is the Ottawa Linux Symposium, in lovely Ottawa, Canada
- the best Linux (kernel) conference in existence ... at least in the northern hemisphere
- OLS 2003 was the fifth OLS
- About 500 attendees, 50 speakers
- More than 150 papers were submitted
- You don’t have to be a kernel guru to attend . . .
- . . . but it helps

How does it work

- the conference is four days long
- one hour long talks, with 30 minutes of break between talks
- about six talks to a day
- four talks at the same time - choose which one you want to hear
- extra activities: reception, sponsor dinner, keynote speech and party
- world's largest pgp key signing party!

How does it REALLY work

- the conference hall is a long room, strawn with sofas and chairs
- computers, cables, switches, wireless access points
- tens of geeks sitting around hacking on their laptops, talking to each other (sometime in real life, sometime on IRC, sometimes both at the same time)
- “will hack for a reliable wireless signal”
- got a bug? the guy who wrote the code is probably in the room!
- finally a chance to meet the people you’ve been talk to and working with for years . . .
- . . . and drink alcohol in one of Ottawa’s many pubs

Shared Page Tables, by Dave McCracken

What are page tables? page tables are data structures that map linear to physical addresses. There's one page table for each address space (roughly, each process). Linux uses a common three-level page tables implementation that all archs adhere to. It also doubles as hardware page table for most archs.

Shared memory areas mapped in many address spaces can take up more space in page table space than in data space, and shared page tables live in low memory, a scarce resource. Shared page tables give substantial space savings.

shpte - why share page tables?

The problem in a nutshell:

- accounting (page table) overhead for an area mapped by a single process is small
- but overhead for each mapped area grows linearly with number of mappings
- massively mapped areas could use more physical pages memory for page tables (overhead) than data pages

Why share? because we can: pte pages for large shared areas are identical in each address_space.
Don’t forget - pte pages live in lowmem which is a scarce resource on big 32 bit machines.

shpte - kernel structures

- mm_struct - roughly one per process, all memory related information for a process is found here
- vma - a virtually mapped region with the same characteristics
- page struct - for each physical page or memory, there is a ’struct page’ representing that page frame (not page contents!)
  - describes how the page is used
  - has a pointer to struct address_space if it’s mapping data from a file
  - all page structs live in the global mem_map data structure
  - with rmap - has a back pointer (or array of back pointers) to all of the ptes that map the page

shpte - how page mappings happen

When a task creates a shared memory area, via either mmap or shmem calls, we don't allocate pages (or page table entries) for it. A page is only mapped when a task faults trying to access it. The fault code finds the correct vma and pte entry, then finds and maps the page. if necessary the pte page is allocated on the fly.

When many tasks share the same shared memory area, there is only one physical page for each page of the shared memory, but one pte page for each task.

shpte - finding shareable pte pages

During page fault time, if the page fault happens on a shareable or read-only vma, we try to find a shareable pte page:

- vma must be shareable, must span entire pte page
- walk address_space chain of vmas looking for one compatible with this vma for sharing
- check the pte page for each compatible vma to see if it can be shared
- if it can be shared, share it by making the new pmd point to the “old” pte page, and increase the reference count on the pte page

shpte - complications

- shared page tables pages may need pointers to multiple mm_structs - mm_struct pointer had to be converted to a chain of mm's
- several system calls may modify mappings and require unsharing pte pages - mremap, mprotect, mmap.
- locking changes required - this is a lot less trivial than it sounds. . .
- vm is still being changed and “fine tuned” - shpte requires constant supervision

share page tables design philosophy: better safe then sorry, if not 100% sure that the sharing is correct, unshare it immediately.

shpte - performance

The purpose of shpte is to save lowmem, not performance. But Linus and Andrew Morton wanted to see performance improvments before merging it. . .

- COW improves fork performance by factor of 10
- unsharing costs as much as fork without COW, plus a little extra
- all programs unshare at least 3 pte pages
- small programs only have 3 pte pages
- simple hack is to not do COW for such programs (with only 3 pte pages)

shpte - status

Patch was stable in about mid-november last year, but has since bit-rotted. dmc is still maintaining it, and intends to bring it up to date when 2.7 opens. Muli is hacking on it for fun - right now, it works with 2.6.0-test1(!), but X won't come up. Work In Progress.

More information

- http://www.linuxsymposium.org
- proceedings: http://archive.linuxsymposium.org/ols2003/Proceedings/
- my notes on OLS: http://www.livejournal.com/users/mulix/14525.html
- Orna Agmon’s notes on OLS: http://vipe.technion.ac.il/ ladypine/OLS2003.html