Subject: Re: [RFC][PATCH 2/2] memory checkpoint with swapfiles
Posted by serue on Thu, 14 Jun 2007 16:16:05 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen (hansendc@us.ibm.com):
>
> We have a lot of options with how to do actual checkpointing
> of a process's memory. We have existing interfaces like
> ptrace and /proc/$pid/mem. But, I'm sure everybody wants to
> be able to checkpoint things with the smallest amount of
> downtime possible, and being able to do it incrementally
> is important.
>
> So, I've hacked up the swap code a bit to take requests
> via a syscall (very temporarily) and shoot down pages that
> were previously mapped and put them in swap. If you want
> to checkpoint such a process, all you have to do is figure
> out which virtual address got placed where in swap, and
> you have all of the data that you need to recreate all of
> the anonymous memory that the process had.
>
> This needs quite a few more bits to be actually useful,
> like making sure that only a single container's data gets
> put into the target swapfile, but it does appear to work.

Another thing this will need is a way to create a very quick
copy-on-write copy of the swapfile after the checkpoint. Is
that simple enough to do?

> Is anybody revolted by this approach?
>
> ---
>
> lxc-dave/include/linux/mm.h | 1 +
> lxc-dave/include/linux/ptrace.h | 1 +
> lxc-dave/include/linux/swapops.h | 5 +++++
> lxc-dave/kernel/ptrace.c | 23 ++++++++++++++++++++++
> lxc-dave/mm/memory.c | 35 ++++++++++++++++++++++++++++++++-
> lxc-dave/mm/migrate.c | 5 -----
> lxc-dave/mm/rmap.c | 2 +-
> lxc-dave/mm/swap_state.c | 4 ++++
> lxc-dave/mm/vmscan.c | 35 +++++++++++++++++++++++++++++++-------
> 9 files changed, 97 insertions(+), 14 deletions(-)
>
> diff -puN include/linux/mm.h~add-ptrace-extension include/linux/mm.h
> --- lxc/include/linux/mm.h~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/include/linux/mm.h 2007-06-13 15:24:40.000000000 -0700
> @@ -1129,6 +1129,7 @@ struct page *follow_page(struct vm_area_

> #define FOLL_TOUCH 0x02 /* mark page accessed */
> #define FOLL_GET 0x04 /* do get_page on page */
> #define FOLL_ANON 0x08 /* give ZERO_PAGE if no pgtable */
> +#define FOLL_SWAP 0x10 /* give ZERO_PAGE if no pgtable */
>
> #ifdef CONFIG_PROC_FS
> void vm_stat_account(struct mm_struct *, unsigned long, struct file *, long);
> diff -puN include/linux/ptrace.h~add-ptrace-extension include/linux/ptrace.h
> --- lxc/include/linux/ptrace.h~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/include/linux/ptrace.h 2007-06-13 15:24:40.000000000 -0700
> @@ -26,6 +26,7 @@
> #define PTRACE_GETEVENTMSG 0x4201
> #define PTRACE_GETSIGINFO 0x4202
> #define PTRACE_SETSIGINFO 0x4203
> +#define PTRACE_POKEPTE 0x4204

Hmm, something about poking the pte's, I suppose? But how come this
isn't used anywhere?

> /* options set using PTRACE_SETOPTIONS */
> #define PTRACE_O_TRACESYSGOOD 0x00000001
> diff -puN include/linux/swapops.h~add-ptrace-extension include/linux/swapops.h
> --- lxc/include/linux/swapops.h~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/include/linux/swapops.h 2007-06-13 15:24:40.000000000 -0700
> @@ -12,6 +12,11 @@
> #define SWP_TYPE_SHIFT(e) (sizeof(e.val) * 8 - MAX_SWAPFILES_SHIFT)
> #define SWP_OFFSET_MASK(e) ((1UL << SWP_TYPE_SHIFT(e)) - 1)
>
> +static inline int is_swap_pte(pte_t pte)
> +{
> + return !pte_none(pte) && !pte_present(pte) && !pte_file(pte);
> +}
> +
> /*
>   * Store a type+offset into a swp_entry_t in an arch-independent format
>   */
> diff -puN kernel/ptrace.c~add-ptrace-extension kernel/ptrace.c
> --- lxc/kernel/ptrace.c~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/kernel/ptrace.c 2007-06-13 15:24:40.000000000 -0700
> @@ -448,6 +448,29 @@ struct task_struct *ptrace_get_task_stru
> }
>
> #ifndef __ARCH_SYS_PTRACE
> +asmlinkage long sys_hackery(long data, long pid, long addr)
> +{
> + int ret = 0;
> + int poke_process_pte(struct task_struct *tsk, unsigned long addr,
> +   pte_t *pte_state);

Odd place to put a prototype :) Were you planning on passing this in
as a fn argument at some point?

> + pte_t pte_state;
> + struct task_struct *child;
> +
> + child = find_task_by_pid(pid);
> + if (child)
> + get_task_struct(child);

Does this count on the process having been placed in the freezer first
through some other mechanism? Or is it safe on its own?

> + ret = poke_process_pte(child, addr, &pte_state);
> + if (ret)
> + goto out;
> + ret = copy_to_user((void *)data,
> + &pte_state,
> + sizeof(pte_state));
> +out:
> + if (child)
> + put_task_struct(child);
> + return ret;
> +}
> +
> asmlinkage long sys_ptrace(long request, long pid, long addr, long data)
> {
> struct task_struct *child;
> diff -puN mm/memory.c~add-ptrace-extension mm/memory.c
> --- lxc/mm/memory.c~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/mm/memory.c 2007-06-13 15:25:17.000000000 -0700
> @@ -941,8 +941,19 @@ struct page *follow_page(struct vm_area_
> goto out;
>
> pte = *ptep;
> - if (!pte_present(pte))
> + if (!pte_present(pte)) {
> + /*
> + * We should probably clean the actual entry up
> + * a bit, but this will do for now
> + */
> + if (is_swap_pte(pte) && (flags & FOLL_SWAP))
> + page = (struct page *)ptep;
> + goto unlock;
> + }
> + if (flags & FOLL_SWAP) {
> + page = NULL;

```
>    goto unlock;
> + }
>   if ((flags & FOLL_WRITE) && !pte_write(pte))
>    goto unlock;
>   page = vm_normal_page(vma, address, pte);
> @@ -2684,6 +2695,28 @@ int in_gate_area_no_task(unsigned long a
>
>  #endif /* __HAVE_ARCH_GATE_AREA */
>
> +int try_to_put_page_in_swap(struct page *page);
> +
> +int poke_process_pte(struct task_struct *tsk, unsigned long addr,
> +          pte_t *pte_state)
> +{
> + struct page *page;
> + struct vm_area_struct *vma;
> +
> + vma = find_vma(tsk->mm, addr);
> + if (!vma)
> +  return -EINVAL;
> + page = follow_page(vma, addr, FOLL_GET);
> + if (!page)
> +  return -EINVAL;
> + try_to_put_page_in_swap(page);
> + put_page(page);
> + page = follow_page(vma, addr, FOLL_SWAP);
> + if (page)
> +  *pte_state = *(pte_t *)page;
> + return 0;
> +}
> +
>  /*
>   * Access another process' address space.
>   * Source/target buffer must be kernel space,
> diff -puN mm/migrate.c~add-ptrace-extension mm/migrate.c
> --- lxc/mm/migrate.c~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/mm/migrate.c 2007-06-13 15:24:40.000000000 -0700
> @@ -115,11 +115,6 @@ int putback_lru_pages(struct list_head *
>   return count;
>  }
>
> -static inline int is_swap_pte(pte_t pte)
> -{
> - return !pte_none(pte) && !pte_present(pte) && !pte_file(pte);
> -}
> -
>  /*
>   * Restore a potential migration pte to a working pte entry
```

>   */
> diff -puN mm/rmap.c~add-ptrace-extension mm/rmap.c
> --- lxc/mm/rmap.c~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/mm/rmap.c 2007-06-13 15:24:40.000000000 -0700
> @@ -795,7 +795,7 @@ static void try_to_unmap_cluster(unsigne
>   pte_unmap_unlock(pte - 1, ptl);
> }
>
> -static int try_to_unmap_anon(struct page *page, int migration)
> +int try_to_unmap_anon(struct page *page, int migration)
> {
>   struct anon_vma *anon_vma;
>   struct vm_area_struct *vma;
> diff -puN mm/swap_state.c~add-ptrace-extension mm/swap_state.c
> --- lxc/mm/swap_state.c~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/mm/swap_state.c 2007-06-13 15:24:40.000000000 -0700
> @@ -128,6 +128,10 @@ void __delete_from_swap_cache(struct pag
>   BUG_ON(PageWriteback(page));
>   BUG_ON(PagePrivate(page));
>
> + if (printk_ratelimit()) {
> + printk("%s(%p)\n", __func__, page);
> + dump_stack();
> + }
>   radix_tree_delete(&swapper_space.page_tree, page_private(page));
>   set_page_private(page, 0);
>   ClearPageSwapCache(page);
> diff -puN mm/vmscan.c~add-ptrace-extension mm/vmscan.c
> --- lxc/mm/vmscan.c~add-ptrace-extension 2007-06-13 15:24:40.000000000 -0700
> +++ lxc-dave/mm/vmscan.c 2007-06-13 15:24:40.000000000 -0700
> @@ -611,19 +611,40 @@ static unsigned long shrink_page_list(st
>
> int try_to_put_page_in_swap(struct page *page)
> {
> -
> - get_page(page);

Ok, so this is called with page's refcount already inc'ed by the
follow_page(), and the caller also put's the page? Should the
fact that a ref to page should be held and put by caller be
commented above, or is that pretty obvious to anyone who would
mess with this file?

thanks,
-serge

> + int ret = 0;
> + struct writeback_control wbc = {

> + .sync_mode = WB_SYNC_NONE,
> + };
>   if (page_count(page) == 1)
>   /* page was freed from under us. So we are done. */
> -   return -EAGAON;
> +   return -EAGAIN;
>   lock_page(page);
>   if (PageWriteback(page))
>   wait_on_page_writeback(page);
> - try_to_unmap(page, 0);
> - printk("page mapped: %d\n", page_mapped(page));
> + if (!PageAnon(page))
> +   goto unlock;
> + if (!PageSwapCache(page))
> + if (!add_to_swap(page, GFP_ATOMIC))
> +   goto unlock;
> +
> + {
> + /*
> + * This used to be a plain try_to_unmap(), but some
> + * pages were getting into the _file() function with
> + * what I think were null ->mapping pointer and oopsing
> + * on the mapping->mapping_lock.
> + */
> + int try_to_unmap_anon(struct page *page, int migration);
> + ret = try_to_unmap_anon(page, 0);
> + }
> + if (!page_mapped(page)) {
> + swap_writepage(page, &wbc);
> + lock_page(page);
> + wait_on_page_writeback(page);
> + }
> +unlock:
>   unlock_page(page);
> - put_page(page);
> - return 0;
> + return ret;
> }
>
> /*
> _
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.linux-foundation.org