Subject: [RFC PATCH 1/1] swap namespaces: introduce basic, simple swap namespaces (v3)
Posted by serue on Mon, 28 Apr 2008 12:57:02 GMT
View Forum Message <> Reply to Message

One way of doing application memory checkpoint which was suggested
by Dave Hansen was to exploit the swapfile for storing memory, and
just dump the relevant metadata.

To that end, this patch allows a set of tasks to be associated with
one specific swapfile. I called it 'swap namespaces', but it's
actually not a namespace at all, just a filter. The initial swap_ns
is null. In this swap_ns multiple swapfiles can be used. In a child
swap_ns, only one swapfile can be swapon'd. A swapfile can only be
active in one swap_ns. When you clone a new swap_ns, you begin with
no swapfile.

You can't unshare a swap_ns, you must use clone. You must also
CLONE_NEWIPC and not CLONE_VM.

This patch doesn't address the limited number of swapfiles per
system, which I imagine will become a tremendous shortcoming for
machines intended to run large number of c/r jobs.

This patch is based on top of Cedric's clone64 patchset so as
to have a free CLONE_SWAPNS flag, so you pass CLONE_SWAPNS in
through cloneflags_hi.

One way to test is as follows:

```
        login 1                     login 2
==========================================================================
dd if=/dev/zero of=/s1 bs=1M \
 count=512
dd if=/dev/zero of=/s2 bs=1M \
 count=512
strings /s2
 (you see only the swap signature)
clone64(CLONE_NEWSWAP|CLONE_NEWIPC)
        clone64(CLONE_NEWSWAP|CLONE_NEWIPC)
swapon /s1
                            swapon /s2
cd $LTPDIR
runltp -l out1 > txt1 2>&1
(let ltp run...)
exit
                            exit
```

Now you can notice the following:

The output of strings /s2 contains no memory from the ltp run.

In ltp the following tests failed in addition to tests on an
unpatched kernel, or a patched kernel but not in a swap_ns:

swapon01
swapon02
swapoff01
swapoff02
mem01

The first four are because you cannot swapon a new swapfile,
since you already have /s1 swapped on.
The last one, if you look at your console, is because mem01 was
killed by oom.

As I often note with patchsets like these, I don't expect this
code or even the design to go upstream, although the patch appears
to work. Rather this posting is mainly to collect feedback and
discussion.

thanks,
-serge

>From b2a5b0cd72fd8adcb58343a6f3ef6b5340ea284a Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Fri, 18 Apr 2008 16:46:10 -0400
Subject: [RFC PATCH 1/1] swap namespaces: introduce basic, simple swap namespaces (v3)

By default processes are in the "init" swapns, which is NULL.
By default, all swapfiles are in the init swapns.

When a task calls clone(CLONE_NEWSWAP), it must also have
no vm in common with any tasks in another swap_namespace.
Its nsproxy->swap_namespace is then a structure containing
an 'int type', which defaults to -1.

A task in a non-init swapns can only have one swapfile.
After swapon, nsproxy->swap_namespace->type will be the
index into the swaplist. The corresponding swap_info_struct
will have swap_namespace pointing to the new swap_namespace.
(For any swapfiles which were swapon()d in the init swapns,
it is NULL).

get_swap_page now takes an argument, the page for which it
is finding a swap page. We use the anonvma to find an mm
to which the page belongs, use the mm->owner to find the

owning task, and use that to find the swap_namespace. If
it exists, then we use its swapfile to find the swap page.

Note:
shmem is a problem at checkpoint - any swapped pages from
an mmap on a shmem file will be in the init swap ns. So
we'll need (1) to copy over any tmpfs files for which there
are open fds, and (2) fine a way to detect mmaped tmpfs files
which no longer have an open fd.

Changelog:
 Apr 24: small CONFIG_SWAP_NS=n fixes
 Apr 23: fix wrongly inverted check or CLONE_VM
 Apr 23: use u64 for clone flags (oops)
 Apr 21: force CLONE_NEWIPC if clone(CLONE_NEWSWAP)
 Apr 18: make sure to filter out private /proc/swaps
  entries when viewing from the init namespace.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 include/linux/nsproxy.h | 2 +
 include/linux/rmap.h | 6 ++
 include/linux/sched.h | 1 +
 include/linux/swap.h | 6 +-
 include/linux/swap_namespace.h | 68 ++++++++++++++++++
 init/Kconfig | 8 ++
 kernel/fork.c | 21 +++++
 kernel/nsproxy.c | 14 +++-
 mm/Makefile | 2 +-
 mm/rmap.c | 4 +-
 mm/shmem.c | 2 +-
 mm/swap_namespace.c | 84 ++++++++++++++++++++++
 mm/swap_state.c | 2 +-
 mm/swapfile.c | 174 +++-
 14 files changed, 385 insertions(+), 9 deletions(-)
 create mode 100644 include/linux/swap_namespace.h
 create mode 100644 mm/swap_namespace.c

diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index 5395e8c..8e2490a 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -8,6 +8,7 @@ struct mnt_namespace;
 struct uts_namespace;
 struct ipc_namespace;
 struct pid_namespace;
+struct swap_namespace;

```
 /*
  * A structure to contain pointers to all per-process
@@ -29,6 +30,7 @@ struct nsproxy {
  struct pid_namespace *pid_ns;
  struct user_namespace *user_ns;
  struct net       *net_ns;
+ struct swap_namespace *swap_ns;
 };
 extern struct nsproxy init_nsproxy;

diff --git a/include/linux/rmap.h b/include/linux/rmap.h
index 1383692..48024db 100644
--- a/include/linux/rmap.h
+++ b/include/linux/rmap.h
@@ -31,6 +31,9 @@ struct anon_vma {

 #ifdef CONFIG_MMU

+extern struct anon_vma *page_lock_anon_vma(struct page *page);
+extern void page_unlock_anon_vma(struct anon_vma *anon_vma);
+
 extern struct kmem_cache *anon_vma_cachep;

 static inline struct anon_vma *anon_vma_alloc(void)
@@ -111,6 +114,9 @@ int page_mkclean(struct page *);

 #else /* !CONFIG_MMU */

+#define page_lock_anon_vma(p) (NULL)
+#define page_unlock_anon_vma(p) do {} while (0)
+
 #define anon_vma_init()  do {} while (0)
 #define anon_vma_prepare(vma) (0)
 #define anon_vma_link(vma) do {} while (0)
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 90130b7..5d59c2f 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -28,6 +28,7 @@
 #define CLONE_NEWPID  0x20000000 /* New pid namespace */
 #define CLONE_NEWNET  0x40000000 /* New network namespace */
 #define CLONE_IO  0x80000000 /* Clone io context */
+#define CLONE_NEWSWAP  0x0000000100000000ULL

 /*
  * Scheduling policies
diff --git a/include/linux/swap.h b/include/linux/swap.h
index 0b33776..e0e8f3d 100644
```

```
--- a/include/linux/swap.h
+++ b/include/linux/swap.h
@@ -7,6 +7,7 @@
 #include <linux/list.h>
 #include <linux/memcontrol.h>
 #include <linux/sched.h>
+#include <linux/swap_namespace.h>

 #include <asm/atomic.h>
 #include <asm/page.h>
@@ -134,6 +135,7 @@ enum {
  */
 struct swap_info_struct {
  unsigned int flags;
+ struct swap_namespace *swap_namespace;
  int prio;   /* swap priority */
  struct file *swap_file;
  struct block_device *bdev;
@@ -239,7 +241,7 @@ extern struct page *swapin_readahead(swp_entry_t, gfp_t,
 extern long total_swap_pages;
 extern unsigned int nr_swapfiles;
 extern void si_swapinfo(struct sysinfo *);
-extern swp_entry_t get_swap_page(void);
+extern swp_entry_t get_swap_page(struct page *page);
 extern swp_entry_t get_swap_page_of_type(int);
 extern int swap_duplicate(swp_entry_t);
 extern int valid_swaphandles(swp_entry_t, unsigned long *);
@@ -342,7 +344,7 @@ static inline int remove_exclusive_swap_page(struct page *p)
  return 0;
 }

-static inline swp_entry_t get_swap_page(void)
+static inline swp_entry_t get_swap_page(struct page *page)
 {
  swp_entry_t entry;
  entry.val = 0;
diff --git a/include/linux/swap_namespace.h b/include/linux/swap_namespace.h
new file mode 100644
index 0000000..3839d58
--- /dev/null
+++ b/include/linux/swap_namespace.h
@@ -0,0 +1,68 @@
+#ifndef _LINUX_SWAP_NS_H
+#define _LINUX_SWAP_NS_H
+
+#include <linux/list.h>
+#include <linux/kref.h>
+#include <linux/err.h>
```

```
+#include <linux/sched.h>
+#include <asm/page.h>
+
+/*
+ * Practically, we store both type and si because both are useful.
+ * During get_swap_page(), we use type==-1 to decide whether there
+ * is a swapfile.
+ * During swapon/swapoff, we use si==NULL.  This is because there
+ * are certain times - especially during swapoff - where we drop the
+ * swap_lock after removing the swapfile, only to find we couldn't
+ * remove the swapfile and need to reactivate it.  Using si==NULL
+ * for the check here allows us to prevent a racing swapon and
+ * swapoff during this window.
+ */
+struct swap_namespace {
+ struct kref  kref;
+ int   type;  /* index into swap_list */
+ struct swap_info_struct *si;
+ /*
+  * ns->lock can only be held *under* swap_lock
+  */
+ spinlock_t  lock;
+};
+
+#ifdef CONFIG_SWAP_NS
+extern void free_swap_ns(struct kref *kref);
+extern struct swap_namespace *copy_swap_ns(u64 flags,
+        struct swap_namespace *ns);
+extern struct swap_namespace *vma_page_to_swapns(struct page *page);
+#else
+static inline void free_swap_ns(struct kref *kref)
+{
+}
+
+static inline struct swap_namespace *copy_swap_ns(u64 flags,
+       struct swap_namespace *ns)
+{
+ /* no namespaces so we don't copy it, just inc the refcount */
+ if (flags&CLONE_NEWSWAP)
+  return ERR_PTR(-EINVAL);
+ return NULL;
+}
+
+static inline struct swap_namespace *vma_page_to_swapns(struct page *page)
+{
+ return NULL;
+}
+#endif
```

```
+
+static inline struct swap_namespace *get_swap_ns(struct swap_namespace *swapns)
+{
+ if (swapns)
+  kref_get(&swapns->kref);
+ return swapns;
+}
+
+static inline void put_swap_ns(struct swap_namespace *swapns)
+{
+ if (swapns)
+  kref_put(&swapns->kref, free_swap_ns);
+}
+#endif
diff --git a/init/Kconfig b/init/Kconfig
index 27b1660..06e3d1e 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -478,6 +478,14 @@ config PID_NS
    Unless you want to work with an experimental feature
    say N here.

+config SWAP_NS
+ bool "Swap Namespaces (EXPERIMENTAL)"
+ default n
+ depends on SWAP && NAMESPACES && EXPERIMENTAL && MMU
+ select MM_OWNER
+ help
+   give some help.
+
 config BLK_DEV_INITRD
  bool "Initial RAM filesystem and RAM disk (initramfs/initrd) support"
  depends on BROKEN || !FRV
diff --git a/kernel/fork.c b/kernel/fork.c
index 8f26d44..1ccf600 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -994,6 +994,20 @@ static void rt_mutex_init_task(struct task_struct *p)
 #endif
 }

+/*
+ * tasks in different swap namespaces cannot share any vm.
+ */
+static int check_swapns_clone(u64 flags)
+{
+ if (!(flags & CLONE_NEWSWAP))
+  return 0;
```

```
+ if (flags & CLONE_VM)
+  return -EINVAL;
+ if (!(flags & CLONE_NEWIPC))
+  return -EINVAL;
+ return 0;
+}
+
 #ifdef CONFIG_MM_OWNER
 void mm_init_owner(struct mm_struct *mm, struct task_struct *p)
 {
@@ -1041,6 +1055,13 @@ static struct task_struct *copy_process(u64 clone_flags,
  if ((clone_flags & CLONE_SIGHAND) && !(clone_flags & CLONE_VM))
   return ERR_PTR(-EINVAL);

+ /*
+  * Can't fork into a new swap namespace while sharing vm
+  */
+ retval = check_swapns_clone(clone_flags);
+ if (retval)
+  return ERR_PTR(retval);
+
  retval = security_task_create(clone_flags);
  if (retval)
   goto fork_out;
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index ded928d..fab101e 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -22,6 +22,7 @@
 #include <linux/pid_namespace.h>
 #include <net/net_namespace.h>
 #include <linux/ipc_namespace.h>
+#include <linux/swap_namespace.h>

 static struct kmem_cache *nsproxy_cachep;

@@ -93,8 +94,17 @@ static struct nsproxy *create_new_namespaces(u64 flags,
   goto out_net;
  }

+ new_nsp->swap_ns = copy_swap_ns(flags, tsk->nsproxy->swap_ns);
+ if (IS_ERR(new_nsp->swap_ns)) {
+  err = PTR_ERR(new_nsp->swap_ns);
+  goto out_swap;
+ }
+
  return new_nsp;
```

```
+out_swap:
+ if (new_nsp->net_ns)
+  put_net(new_nsp->net_ns);
 out_net:
  if (new_nsp->user_ns)
   put_user_ns(new_nsp->user_ns);
@@ -131,7 +141,8 @@ int copy_namespaces(u64 flags, struct task_struct *tsk)
  get_nsproxy(old_ns);

  if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
-    CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET)))
+    CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET |
+    CLONE_NEWSWAP)))
   return 0;

  if (!capable(CAP_SYS_ADMIN)) {
@@ -183,6 +194,7 @@ void free_nsproxy(struct nsproxy *ns)
  if (ns->user_ns)
   put_user_ns(ns->user_ns);
  put_net(ns->net_ns);
+ put_swap_ns(ns->swap_ns);
  kmem_cache_free(nsproxy_cachep, ns);
 }

diff --git a/mm/Makefile b/mm/Makefile
index 18c143b..40c7224 100644
--- a/mm/Makefile
+++ b/mm/Makefile
@@ -11,7 +11,7 @@ obj-y   := bootmem.o filemap.o mempool.o oom_kill.o fadvise.o \
       maccess.o page_alloc.o page-writeback.o pdflush.o \
       readahead.o swap.o truncate.o vmscan.o \
       prio_tree.o util.o mmzone.o vmstat.o backing-dev.o \
-      page_isolation.o $(mmu-y)
+      page_isolation.o swap_namespace.o $(mmu-y)

 obj-$(CONFIG_PROC_PAGE_MONITOR) += pagewalk.o
 obj-$(CONFIG_BOUNCE) += bounce.o
diff --git a/mm/rmap.c b/mm/rmap.c
index 6901c6d..f18528e 100644
--- a/mm/rmap.c
+++ b/mm/rmap.c
@@ -156,7 +156,7 @@ void __init anon_vma_init(void)
  * Getting a lock on a stable anon_vma from a page off the LRU is
  * tricky: page_lock_anon_vma rely on RCU to guard against the races.
  */
-static struct anon_vma *page_lock_anon_vma(struct page *page)
+struct anon_vma *page_lock_anon_vma(struct page *page)
 {
```

```
  struct anon_vma *anon_vma;
  unsigned long anon_mapping;
@@ -176,7 +176,7 @@ out:
  return NULL;
 }

-static void page_unlock_anon_vma(struct anon_vma *anon_vma)
+void page_unlock_anon_vma(struct anon_vma *anon_vma)
 {
  spin_unlock(&anon_vma->lock);
  rcu_read_unlock();
diff --git a/mm/shmem.c b/mm/shmem.c
index e2a6ae1..8573094 100644
--- a/mm/shmem.c
+++ b/mm/shmem.c
@@ -1021,7 +1021,7 @@ static int shmem_writepage(struct page *page, struct
writeback_control *wbc)
   * want to check if there's a redundant swappage to be discarded.
   */
  if (wbc->for_reclaim)
-  swap = get_swap_page();
+  swap = get_swap_page(page);
  else
   swap.val = 0;

diff --git a/mm/swap_namespace.c b/mm/swap_namespace.c
new file mode 100644
index 0000000..5f5ff3a
--- /dev/null
+++ b/mm/swap_namespace.c
@@ -0,0 +1,84 @@
+/*
+ *  Copyright (C) 2008 IBM Corporation
+ *
+ *  Author: Serge Hallyn <serue@us.ibm.com>
+ *
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License as
+ *  published by the Free Software Foundation, version 2 of the
+ *  License.
+ */
+
+#include <linux/module.h>
+#include <linux/swap_namespace.h>
+#include <linux/swap.h>
+#include <linux/version.h>
+#include <linux/err.h>
+#include <linux/sched.h>
```

```
+#include <linux/nsproxy.h>
+#include <linux/rmap.h>
+
+#ifdef CONFIG_SWAP_NS
+extern void free_swapinfo_fromnsput(struct swap_info_struct *si);
+
+void free_swap_ns(struct kref *kref)
+{
+ struct swap_namespace *ns;
+
+ ns = container_of(kref, struct swap_namespace, kref);
+ if (ns->si)
+  free_swapinfo_fromnsput(ns->si);
+ kfree(ns);
+}
+
+struct swap_namespace *clone_swapns(struct swap_namespace *old_ns)
+{
+ struct swap_namespace *new_ns;
+
+ new_ns = kmalloc(sizeof(struct swap_namespace), GFP_KERNEL);
+ if (!new_ns)
+  return ERR_PTR(-ENOMEM);
+
+ kref_init(&new_ns->kref);
+ spin_lock_init(&new_ns->lock);
+ new_ns->si = NULL;
+ new_ns->type = -1;
+ return new_ns;
+}
+
+struct swap_namespace *copy_swap_ns(u64 flags, struct swap_namespace *ns)
+{
+ struct swap_namespace *new_ns;
+
+ get_swap_ns(ns);
+
+ if (!(flags & CLONE_NEWSWAP))
+  return ns;
+
+ new_ns = clone_swapns(ns);
+ put_swap_ns(ns);
+ return new_ns;
+}
+
+struct swap_namespace *vma_page_to_swapns(struct page *page)
+{
+ struct swap_namespace *ns = NULL;
```

```
+ struct anon_vma *anon_vma;
+ struct vm_area_struct *vma;
+ struct mm_struct *mm;
+ struct task_struct *task;
+
+ anon_vma = page_lock_anon_vma(page);
+ if (!anon_vma)
+  return NULL;
+
+ /* we are under rcu_read_lock from page_lock_anon_vma */
+ vma = list_first_entry(&anon_vma->head, struct vm_area_struct,
+    anon_vma_node);
+ mm = vma->vm_mm;
+ task = rcu_dereference(mm->owner);
+ ns = task_nsproxy(task)->swap_ns;
+ page_unlock_anon_vma(anon_vma);
+ return ns;
+}
+#endif
diff --git a/mm/swap_state.c b/mm/swap_state.c
index d8aadaf..182ef26 100644
--- a/mm/swap_state.c
+++ b/mm/swap_state.c
@@ -129,7 +129,7 @@ int add_to_swap(struct page * page, gfp_t gfp_mask)
  BUG_ON(!PageUptodate(page));

  for (;;) {
-  entry = get_swap_page();
+  entry = get_swap_page(page);
   if (!entry.val)
    return 0;

diff --git a/mm/swapfile.c b/mm/swapfile.c
index bd1bb59..1c302b4 100644
--- a/mm/swapfile.c
+++ b/mm/swapfile.c
@@ -28,6 +28,8 @@
 #include <linux/capability.h>
 #include <linux/syscalls.h>
 #include <linux/memcontrol.h>
+#include <linux/nsproxy.h>
+#include <linux/swap_namespace.h>

 #include <asm/pgtable.h>
 #include <asm/tlbflush.h>
@@ -172,12 +174,46 @@ no_page:
  return 0;
 }
```

```
-swp_entry_t get_swap_page(void)
+swp_entry_t swap_page_in_swapfile(int type)
+{
+ struct swap_info_struct *si = swap_info + type;
+ pgoff_t offset;
+
+ if (type == -1)
+  goto noswap;
+
+ spin_lock(&swap_lock);
+ if (si->pages - si->inuse_pages < 1)
+  goto noswap;
+ if (!si->highest_bit)
+  goto noswap;
+ if (!(si->flags & SWP_WRITEOK))
+  goto noswap;
+ nr_swap_pages--;
+ offset = scan_swap_map(si);
+ if (offset) {
+  spin_unlock(&swap_lock);
+  return swp_entry(type, offset);
+ }
+
+ nr_swap_pages++;
+noswap:
+ spin_unlock(&swap_lock);
+ return (swp_entry_t) {0};
+}
+
+swp_entry_t get_swap_page(struct page *page)
 {
  struct swap_info_struct *si;
  pgoff_t offset;
  int type, next;
  int wrapped = 0;
+ struct swap_namespace *swap_namespace;
+
+ swap_namespace = vma_page_to_swapns(page);
+ if (swap_namespace) {
+  return swap_page_in_swapfile(swap_namespace->type);
+ }

  spin_lock(&swap_lock);
  if (nr_swap_pages <= 0)
@@ -193,6 +229,8 @@ swp_entry_t get_swap_page(void)
    wrapped++;
   }
```

```
+  if (si->swap_namespace)
+   continue;
   if (!si->highest_bit)
    continue;
   if (!(si->flags & SWP_WRITEOK))
@@ -1203,6 +1241,102 @@ int page_queue_congested(struct page *page)
 }
 #endif

+#ifdef CONFIG_SWAP_NS
+static void unlink_si_locked(struct swap_info_struct *si, int type, int prev)
+{
+ if (prev < 0)
+  swap_list.head = si->next;
+ else
+  swap_info[prev].next = si->next;
+ if (type == swap_list.next) {
+  /* just pick something that's safe... */
+  swap_list.next = swap_list.head;
+ }
+ nr_swap_pages -= si->pages;
+ total_swap_pages -= si->pages;
+ si->flags &= ~SWP_WRITEOK;
+}
+
+static int find_and_unlink_si_locked(struct swap_info_struct *si)
+{
+ int prev = -1, type;
+ struct swap_info_struct *tmp;
+
+ for (type = swap_list.head; type >= 0; type = swap_info[type].next) {
+  tmp = swap_info+type;
+  if (tmp == si)
+   break;
+  prev = type;
+ }
+ if (type < 0)
+  return 0;
+
+ unlink_si_locked(si, type, prev);
+
+ return 1;
+}
+
+static struct file *find_and_zero_si_locked(struct swap_info_struct *si)
+{
+ struct file *swap_file;
```

```
+
+ if (!find_and_unlink_si_locked(si))
+  return NULL;
+
+ swap_file = si->swap_file;
+ si->swap_file = NULL;
+ si->max = 0;
+ si->swap_map = NULL;
+ si->flags = 0;
+
+ return swap_file;
+}
+
+static void clear_swap_file(struct swap_info_struct *si, struct file *swap_file)
+{
+ struct inode *inode;
+ struct address_space *mapping = swap_file->f_mapping;
+
+ inode = mapping->host;
+ if (S_ISBLK(inode->i_mode)) {
+  struct block_device *bdev = I_BDEV(inode);
+  set_blocksize(bdev, si->old_block_size);
+  bd_release(bdev);
+ } else {
+  mutex_lock(&inode->i_mutex);
+  inode->i_flags &= ~S_SWAPFILE;
+  mutex_unlock(&inode->i_mutex);
+ }
+ filp_close(swap_file, NULL);
+}
+/*
+ * free_swapinfo_fromnsput
+ * Called when the last task in a swap ns exits, puts the
+ * swapinfo, and no other swap namespaces had references
+ * to the swapinfo.
+ * Since we were freeing the swap_ns, we're not holding the
+ * ns->lock, so it's safe to grab swap_lock here.
+ */
+void free_swapinfo_fromnsput(struct swap_info_struct *si)
+{
+ unsigned short *swap_map;
+ struct file *swap_file;
+
+ mutex_lock(&swapon_mutex);
+ spin_lock(&swap_lock);
+ swap_map = si->swap_map;
+ swap_file = find_and_zero_si_locked(si);
+ spin_unlock(&swap_lock);
```

```
+ mutex_unlock(&swapon_mutex);
+
+ if (!swap_file)
+  return;
+
+ vfree(swap_map);
+ clear_swap_file(si, swap_file);
+}
+#endif
+
 asmlinkage long sys_swapoff(const char __user * specialfile)
 {
  struct swap_info_struct * p = NULL;
@@ -1213,6 +1347,7 @@ asmlinkage long sys_swapoff(const char __user * specialfile)
  char * pathname;
  int i, type, prev;
  int err;
+ struct swap_namespace *swap_namespace = current->nsproxy->swap_ns;

  if (!capable(CAP_SYS_ADMIN))
   return -EPERM;
@@ -1244,6 +1379,11 @@ asmlinkage long sys_swapoff(const char __user * specialfile)
   spin_unlock(&swap_lock);
   goto out_dput;
  }
+ if (swap_namespace && type != swap_namespace->type) {
+  err = -EPERM;
+  spin_unlock(&swap_lock);
+  goto out_dput;
+ }
  if (!security_vm_enough_memory(p->pages))
   vm_unacct_memory(p->pages);
  else {
@@ -1263,6 +1403,10 @@ asmlinkage long sys_swapoff(const char __user * specialfile)
  nr_swap_pages -= p->pages;
  total_swap_pages -= p->pages;
  p->flags &= ~SWP_WRITEOK;
+ if (swap_namespace) {
+  swap_namespace->si = NULL;
+  swap_namespace->type = -1;
+ }
  spin_unlock(&swap_lock);

  current->flags |= PF_SWAPOFF;
@@ -1282,6 +1426,10 @@ asmlinkage long sys_swapoff(const char __user * specialfile)
   swap_info[prev].next = p - swap_info;
   nr_swap_pages += p->pages;
   total_swap_pages += p->pages;
```

```
+  if (swap_namespace) {
+   swap_namespace->si = p;
+   swap_namespace->type = type;
+  }
   p->flags |= SWP_WRITEOK;
   spin_unlock(&swap_lock);
   goto out_dput;
@@ -1339,6 +1487,7 @@ static void *swap_start(struct seq_file *swap, loff_t *pos)
  struct swap_info_struct *ptr = swap_info;
  int i;
  loff_t l = *pos;
+ struct swap_namespace *swap_namespace = current->nsproxy->swap_ns;

  mutex_lock(&swapon_mutex);

@@ -1348,6 +1497,10 @@ static void *swap_start(struct seq_file *swap, loff_t *pos)
  for (i = 0; i < nr_swapfiles; i++, ptr++) {
   if (!(ptr->flags & SWP_USED) || !ptr->swap_map)
    continue;
+  if (swap_namespace && ptr->swap_namespace != swap_namespace)
+   continue;
+  if (!swap_namespace && ptr->swap_namespace)
+   continue;
   if (!--l)
    return ptr;
  }
@@ -1359,6 +1512,7 @@ static void *swap_next(struct seq_file *swap, void *v, loff_t *pos)
 {
  struct swap_info_struct *ptr;
  struct swap_info_struct *endptr = swap_info + nr_swapfiles;
+ struct swap_namespace *swap_namespace = current->nsproxy->swap_ns;

  if (v == SEQ_START_TOKEN)
   ptr = swap_info;
@@ -1370,6 +1524,10 @@ static void *swap_next(struct seq_file *swap, void *v, loff_t *pos)
  for (; ptr < endptr; ptr++) {
   if (!(ptr->flags & SWP_USED) || !ptr->swap_map)
    continue;
+  if (swap_namespace && ptr->swap_namespace != swap_namespace)
+   continue;
+  if (!swap_namespace && ptr->swap_namespace)
+   continue;
   ++*pos;
   return ptr;
  }
@@ -1459,10 +1617,17 @@ asmlinkage long sys_swapon(const char __user * specialfile, int
swap_flags)
  struct page *page = NULL;
```

```
  struct inode *inode = NULL;
  int did_down = 0;
+ struct swap_namespace *swap_namespace = current->nsproxy->swap_ns;

  if (!capable(CAP_SYS_ADMIN))
   return -EPERM;
  spin_lock(&swap_lock);
+ if (swap_namespace && swap_namespace->si) {
+  /* only one swapfile for non-init swap_namespaces */
+  error = -EPERM;
+  spin_unlock(&swap_lock);
+  goto out;
+ }
  p = swap_info;
  for (type = 0 ; type < nr_swapfiles ; type++,p++)
   if (!(p->flags & SWP_USED))
@@ -1490,6 +1655,8 @@ asmlinkage long sys_swapon(const char __user * specialfile, int
swap_flags)
  } else {
   p->prio = --least_priority;
  }
+ if (swap_namespace)
+  swap_namespace->si = p;
  spin_unlock(&swap_lock);
  name = getname(specialfile);
  error = PTR_ERR(name);
@@ -1672,6 +1839,9 @@ asmlinkage long sys_swapon(const char __user * specialfile, int
swap_flags)

  mutex_lock(&swapon_mutex);
  spin_lock(&swap_lock);
+ p->swap_namespace = swap_namespace;
+ if (swap_namespace)
+  swap_namespace->type = type;
  p->flags = SWP_ACTIVE;
  nr_swap_pages += nr_good_pages;
  total_swap_pages += nr_good_pages;
@@ -1709,6 +1879,8 @@ bad_swap_2:
  spin_lock(&swap_lock);
  swap_map = p->swap_map;
  p->swap_file = NULL;
+ if (swap_namespace)
+  swap_namespace->si = NULL;
  p->swap_map = NULL;
  p->flags = 0;
  if (!(swap_flags & SWAP_FLAG_PREFER))
--
1.5.3.6
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers