Subject: Re: Re: [RFD][PATCH] memcg: Move Usage at Task Move
Posted by serue on Thu, 12 Jun 2008 21:08:12 GMT
View Forum Message <> Reply to Message

Quoting kamezawa.hiroyu@jp.fujitsu.com (kamezawa.hiroyu@jp.fujitsu.com):
> ----- Original Message -----
> >> Just a question:
> >> What happens when a thread (not thread-group-leader) changes its ns by
> >> ns-cgroup ? not-allowed ?
> >
> >I don't quite understand the question. I assume you're asking whether
> >your cgroup, when composed with ns, will refuse a task in cgroup /cg/1/2
> >from being able to
> >
> > mkdir /cg/1/2/3
> > echo $$ > /cg/1/2/3/tasks
> >
> >or
> >
> > unshare(CLONE_NEWNS)
> >
> >which the ns cgroup would allow, and what your cgroup would do in that
> >case. If your question ("not-allowed ?") is about ns cgroup behavior
> >then please rephrase.
>
> Ah, sorry. I'm just curious. (and I should read the code before making
> quiestion.)
>
> Assume a thread group contains threadA, threadB, threadC.
>
> I wanted to ask "Can threadA, and threadB, and threadC
> be in different cgroups ? And if so, how ns cgroup handles it ?"
>
> Maybe I don't understand ns cgroup.

In part yes, but nonetheless a very interesting question when it comes
to composition of cgroups!

Yes, you can have threads in different cgroups. The ns cgroup just
tracks nsproxy unshares. So if you run the attached program and look
around, you'll see the first thread is in /cg/taskpid while the second
one is in /cg/taskpid/secondthreadpid.

Clearly, composing this with a cgroup which needs to keep threads in the
same cgroup becomes problematic!

Interesting :)

-serge

```
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
#include <libgen.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

#include <linux/unistd.h>

#ifndef SYS_unshare
#ifdef __NR_unshare
#define SYS_unshare __NR_unshare
#elif __i386__
#define SYS_unshare 310
#elif __ia64__
#define SYS_unshare 1296
#elif __x86_64__
#define SYS_unshare 272
#elif __s390x__
#define SYS_unshare 303
#elif __powerpc__
#define SYS_unshare 282
#else
#error "unshare not supported on this architecure."
#endif
#endif

#define CSIGNAL         0x000000ff      /* signal mask to be sent at exit */
#define CLONE_VM        0x00000100      /* set if VM shared between processes */
#define CLONE_FS        0x00000200      /* set if fs info shared between processes */
#define CLONE_FILES     0x00000400      /* set if open files shared between processes */
#define CLONE_SIGHAND   0x00000800      /* set if signal handlers and blocked signals shared
*/
#define CLONE_PTRACE    0x00002000      /* set if we want to let tracing continue on the child
too */
#define CLONE_VFORK     0x00004000      /* set if the parent wants the child to wake it up on
mm_release
*/
#define CLONE_PARENT    0x00008000      /* set if we want to have the same parent as the
cloner */
```

```
#define CLONE_THREAD    0x00010000      /* Same thread group? */
#define CLONE_NEWNS     0x00020000      /* New namespace group? */
#define CLONE_SYSVSEM   0x00040000      /* share system V SEM_UNDO semantics */
#define CLONE_SETTLS    0x00080000      /* create a new TLS for the child */
#define CLONE_PARENT_SETTID     0x00100000      /* set the TID in the parent */
#define CLONE_CHILD_CLEARTID    0x00200000      /* clear the TID in the child */
#define CLONE_DETACHED          0x00400000      /* Unused, ignored */
#define CLONE_UNTRACED          0x00800000      /* set if the tracing process can't force
CLONE_PTRACE on
 this clone */
#define CLONE_CHILD_SETTID      0x01000000      /* set the TID in the child */
#define CLONE_STOPPED           0x02000000      /* Start in stopped state */
#define CLONE_NEWUTS            0x04000000      /* New utsname group? */
#define CLONE_NEWIPC            0x08000000      /* New ipcs */
#define CLONE_NEWNUSER          0x10000000      /* New level 2 network namespace */
#define CLONE_NEWPID            0x20000000      /* New pid namespace */

int child2(void *data)
{
 sleep(500);
}

int child1(void *data)
{
 int stacksize = 8*getpagesize();
 void *childstack, *stack = malloc(stacksize);
 unsigned long flags;
 int ret;

 if (!stack) {
  perror("malloc");
  return -1;
 }
 childstack = stack + stacksize;

 flags = CLONE_THREAD | CLONE_VM | CLONE_SIGHAND | CLONE_NEWNS |
CLONE_NEWUTS;
 ret = clone(child2, childstack, flags, NULL);
 if (ret == -1) {
  perror("clone2");
  return -1;
 }

 sleep(500);
}

int main(int argc, char *argv[])
{
```

```
 int stacksize = 4*getpagesize();
 int pid, ret, status;
 void *childstack, *stack = malloc(stacksize);
 unsigned long flags;

 if (!stack) {
  perror("malloc");
  return -1;
 }
 childstack = stack + stacksize;

 flags = CLONE_NEWNS | CLONE_NEWUTS;
 ret = clone(child1, childstack, flags, (void *)argv);
 if (ret == -1) {
  perror("clone");
  return -1;
 }

 pid = ret;
 while ((ret = waitpid(pid, &status, __WALL) != -1)) {
  printf("pid %d, status %d, ret %d\n",
    pid, status, ret);
 };
 printf("pid %d exited with status %d\n", pid, status);
 exit(0);
}
```

_____

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