
Subject: [patch 3/3] add the clone64() and unshare64() syscalls
Posted by [Cedric Le Goater](#) on Thu, 07 Feb 2008 10:31:38 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Cedric Le Goater <clg@fr.ibm.com>

This patch adds 2 new syscalls :

```
long sys_clone64(unsigned long flags_high, unsigned long flags_low,  
unsigned long newsp);
```

```
long sys_unshare64(unsigned long flags_high, unsigned long flags_low);
```

clone64() does not support CLONE_PARENT_SETTID and CLONE_CHILD_CLEARTID
because we would exceed the 6 registers limit of some arches.

This is work in progress but already includes support for x86, x86_64,
x86_64(32), ppc64, ppc64(32), s390x, s390x(31).

ia64 already supports 64bits clone flags through the clone2() syscall.
should we harmonize the name to clone2 ?

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

```
arch/powerpc/kernel/entry_32.S | 8 +++++++  
arch/powerpc/kernel/entry_64.S | 5 +++++  
arch/powerpc/kernel/process.c | 15 ++++++++  
arch/s390/kernel/compat_linux.c | 15 ++++++++  
arch/s390/kernel/compat_wrapper.S | 6 +++++  
arch/s390/kernel/process.c | 15 ++++++++  
arch/s390/kernel/syscalls.S | 2 ++  
arch/x86/ia32/ia32entry.S | 4 +++++  
arch/x86/ia32/sys_ia32.c | 12 ++++++++  
arch/x86/kernel/entry_64.S | 1 +  
arch/x86/kernel/process_32.c | 14 ++++++++  
arch/x86/kernel/process_64.c | 15 ++++++++  
arch/x86/kernel/syscall_table_32.S | 2 ++  
include/asm-powerpc/systbl.h | 2 ++  
include/asm-powerpc/unistd.h | 4 +++-  
include/asm-s390/unistd.h | 4 +++-  
include/asm-x86/unistd_32.h | 2 ++  
include/asm-x86/unistd_64.h | 4 +++++  
include/linux/syscalls.h | 2 ++  
kernel/fork.c | 7 +++++++  
kernel/sys_ni.c | 3 +++  
21 files changed, 140 insertions(+), 2 deletions(-)
```

Index: 2.6.24-mm1/arch/s390/kernel/syscalls.S

--- 2.6.24-mm1.orig/arch/s390/kernel/syscalls.S

+++ 2.6.24-mm1/arch/s390/kernel/syscalls.S

@@ -327,3 +327,5 @@ SYSCALL(sys_utimensat,sys_utimensat,comp

SYSCALL(sys_signalfd,sys_signalfd,compat_sys_signalfd_wrapper)

NI_SYSCALL /* 317 old sys_timer_fd */

SYSCALL(sys_eventfd,sys_eventfd,sys_eventfd_wrapper)

+SYSCALL(sys_clone64,sys_clone64,sys32_clone64)

+SYSCALL(sys_unshare64,sys_unshare64,sys_unshare64_wrapper)

Index: 2.6.24-mm1/arch/x86/kernel/syscall_table_32.S

--- 2.6.24-mm1.orig/arch/x86/kernel/syscall_table_32.S

+++ 2.6.24-mm1/arch/x86/kernel/syscall_table_32.S

@@ -326,3 +326,5 @@ ENTRY(sys_call_table)

.long sys_fallocate

.long sys_timerfd_settime /* 325 */

.long sys_timerfd_gettime

+ .long sys_clone64

+ .long sys_unshare64

Index: 2.6.24-mm1/include/asm-powerpc/systbl.h

--- 2.6.24-mm1.orig/include/asm-powerpc/systbl.h

+++ 2.6.24-mm1/include/asm-powerpc/systbl.h

@@ -314,3 +314,5 @@ SYSCALL_SPU(eventfd)

COMPAT_SYS_SPU(sync_file_range2)

COMPAT_SYS(fallocate)

SYSCALL(subpage_prot)

+PPC_SYS(clone64)

+SYSCALL_SPU(unshare64)

Index: 2.6.24-mm1/include/asm-powerpc/unistd.h

--- 2.6.24-mm1.orig/include/asm-powerpc/unistd.h

+++ 2.6.24-mm1/include/asm-powerpc/unistd.h

@@ -333,10 +333,12 @@

#define __NR_sync_file_range2 308

#define __NR_fallocate 309

#define __NR_subpage_prot 310

+#define __NR_clone64 311

+#define __NR_unshare64 312

#ifdef __KERNEL__

-#define __NR_syscalls 311

+#define __NR_syscalls 313

#define __NR__exit __NR_exit

#define NR_syscalls __NR_syscalls

Index: 2.6.24-mm1/include/asm-s390/unistd.h

```
=====
--- 2.6.24-mm1.orig/include/asm-s390/unistd.h
+++ 2.6.24-mm1/include/asm-s390/unistd.h
@@ -256,7 +256,9 @@
#define __NR_signalfd 316
#define __NR_timerfd 317
#define __NR_eventfd 318
-#define NR_syscalls 319
+#define __NR_clone64 319
+#define __NR_unshare64 320
+#define NR_syscalls 321

/*
 * There are some system calls that are not present on 64 bit, some
Index: 2.6.24-mm1/include/asm-x86/unistd_32.h
```

```
=====
--- 2.6.24-mm1.orig/include/asm-x86/unistd_32.h
+++ 2.6.24-mm1/include/asm-x86/unistd_32.h
@@ -332,6 +332,8 @@
#define __NR_fallocate 324
#define __NR_timerfd_settime 325
#define __NR_timerfd_gettime 326
+#define __NR_clone64 327
+#define __NR_unshare64 328

#ifdef __KERNEL__
```

Index: 2.6.24-mm1/include/asm-x86/unistd_64.h

```
=====
--- 2.6.24-mm1.orig/include/asm-x86/unistd_64.h
+++ 2.6.24-mm1/include/asm-x86/unistd_64.h
@@ -639,6 +639,10 @@ __SYSCALL(__NR_fallocate, sys_fallocate)
__SYSCALL(__NR_timerfd_settime, sys_timerfd_settime)
#define __NR_timerfd_gettime 287
__SYSCALL(__NR_timerfd_gettime, sys_timerfd_gettime)
+#define __NR_clone64 288
+__SYSCALL(__NR_clone64, stub_clone64)
+#define __NR_unshare64 289
+__SYSCALL(__NR_unshare64, sys_unshare64)

#endif __NO_STUBS
```

Index: 2.6.24-mm1/include/linux/syscalls.h

```
=====
--- 2.6.24-mm1.orig/include/linux/syscalls.h
+++ 2.6.24-mm1/include/linux/syscalls.h
@@ -615,6 +615,8 @@ asmlinkage long sys_timerfd_gettime(int
```

```

asmlinkage long sys_eventfd(unsigned int count);
asmlinkage long sys_fallocate(int fd, int mode, loff_t offset, loff_t len);

+asmlinkage long sys_unshare64(unsigned long clone_flags_high, unsigned long
clone_flags_low);
+
int kernel_execve(const char *filename, char *const argv[], char *const envp[]);

```

```
#endif
```

```
Index: 2.6.24-mm1/kernel/sys_ni.c
```

```
=====
```

```
--- 2.6.24-mm1.orig/kernel/sys_ni.c
```

```
+++ 2.6.24-mm1/kernel/sys_ni.c
```

```
@@ -161,3 +161,6 @@ cond_syscall(sys_timerfd_gettime);
```

```
cond_syscall(compat_sys_timerfd_settime);
```

```
cond_syscall(compat_sys_timerfd_gettime);
```

```
cond_syscall(sys_eventfd);
```

```
+
```

```
+cond_syscall(sys_clone64);
```

```
+cond_syscall(sys_unshare64);
```

```
Index: 2.6.24-mm1/arch/x86/kernel/process_32.c
```

```
=====
```

```
--- 2.6.24-mm1.orig/arch/x86/kernel/process_32.c
```

```
+++ 2.6.24-mm1/arch/x86/kernel/process_32.c
```

```
@@ -768,6 +768,20 @@ asmlinkage int sys_clone(struct pt_regs
```

```
return do_fork(clone_flags, newsp, &regs, 0, parent_tidptr, child_tidptr);
```

```
}
```

```
+asmlinkage int sys_clone64(struct pt_regs regs)
```

```
+{
```

```
+ u64 clone_flags;
```

```
+ unsigned long newsp;
```

```
+
```

```
+ clone_flags = ((u64) regs.bx << 32 | regs.cx);
```

```
+ clone_flags &= ~(CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID);
```

```
+
```

```
+ newsp = regs.dx;
```

```
+ if (!newsp)
```

```
+ newsp = regs.sp;
```

```
+ return do_fork(clone_flags, newsp, &regs, 0, NULL, NULL);
```

```
+}
```

```
+
```

```
/*
```

```
* This is trivial, and on the face of it looks like it
```

```
* could equally well be done in user mode.
```

```
Index: 2.6.24-mm1/arch/x86/kernel/process_64.c
```

```
=====
```

```
--- 2.6.24-mm1.orig/arch/x86/kernel/process_64.c
```

```

+++ 2.6.24-mm1/arch/x86/kernel/process_64.c
@@ -772,6 +772,21 @@ sys_clone(unsigned long clone_flags, uns
    return do_fork(clone_flags, newsp, regs, 0, parent_tid, child_tid);
}

+asmlinkage long
+sys_clone64(unsigned long clone_flags_high, unsigned long clone_flags_low,
+ unsigned long newsp, struct pt_regs *regs)
+{
+ u64 clone_flags;
+
+ clone_flags = ((u64) clone_flags_high << 32 | clone_flags_low);
+ clone_flags &= ~(CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID);
+
+ if (!newsp)
+   newsp = regs->sp;
+ return do_fork(clone_flags, newsp, regs, 0, NULL, NULL);
+}
+
+
+/*
+ * This is trivial, and on the face of it looks like it
+ * could equally well be done in user mode.
Index: 2.6.24-mm1/arch/s390/kernel/compat_linux.c
=====
--- 2.6.24-mm1.orig/arch/s390/kernel/compat_linux.c
+++ 2.6.24-mm1/arch/s390/kernel/compat_linux.c
@@ -940,6 +940,21 @@ asmlinkage long sys32_clone(void)
    parent_tidptr, child_tidptr);
}

+asmlinkage long sys32_clone64(void)
+{
+ struct pt_regs *regs = task_pt_regs(current);
+ u64 clone_flags;
+ unsigned long newsp;
+
+ clone_flags = ((u64) (regs->orig_gpr2 & 0xffffffffUL) << 32 | (regs->gprs[3] & 0xffffffffUL));
+ clone_flags &= ~(CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID);
+
+ newsp = regs->gprs[4] & 0x7fffffffUL;
+ if (!newsp)
+   newsp = regs->gprs[15];
+ return do_fork(clone_flags, newsp, regs, 0, NULL, NULL);
+}
+
+/*
+ * 31 bit emulation wrapper functions for sys_fadvise64/fadvise64_64.

```

* These need to rewrite the advise values for POSIX_FADV_{DONTNEED,NOREUSE}

Index: 2.6.24-mm1/arch/s390/kernel/process.c

```
=====
--- 2.6.24-mm1.orig/arch/s390/kernel/process.c
+++ 2.6.24-mm1/arch/s390/kernel/process.c
@@ -323,6 +323,21 @@ asmlinkage long sys_clone(void)
     parent_tidptr, child_tidptr);
 }

+asmlinkage long sys_clone64(void)
+{
+ struct pt_regs *regs = task_pt_regs(current);
+ u64 clone_flags;
+ unsigned long newsp;
+
+ clone_flags = ((u64) regs->orig_gpr2 << 32 | regs->gprs[3]);
+ clone_flags &= ~(CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID);
+
+ newsp = regs->gprs[4];
+ if (!newsp)
+ newsp = regs->gprs[15];
+ return do_fork(clone_flags, newsp, regs, 0, NULL, NULL);
+}
+
+/*
+ * This is trivial, and on the face of it looks like it
+ * could equally well be done in user mode.
```

Index: 2.6.24-mm1/arch/powerpc/kernel/process.c

```
=====
--- 2.6.24-mm1.orig/arch/powerpc/kernel/process.c
+++ 2.6.24-mm1/arch/powerpc/kernel/process.c
@@ -829,6 +829,21 @@ int sys_clone(unsigned long clone_flags,
     return do_fork(clone_flags, usp, regs, 0, parent_tidp, child_tidp);
 }

+int sys_clone64(unsigned long clone_flags_high, unsigned long clone_flags_low,
+ unsigned long usp, unsigned long p4, unsigned long p5,
+ unsigned long p6, struct pt_regs *regs)
+{
+ u64 clone_flags;
+
+ clone_flags = ((u64) clone_flags_high << 32 | clone_flags_low);
+ clone_flags &= ~(CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID);
+
+ CHECK_FULL_REGS(regs);
+ if (usp == 0)
+ usp = regs->gpr[1]; /* stack pointer for child */
+ return do_fork(clone_flags, usp, regs, 0, NULL, NULL);
```

```

+}
+
int sys_fork(unsigned long p1, unsigned long p2, unsigned long p3,
            unsigned long p4, unsigned long p5, unsigned long p6,
            struct pt_regs *regs)
Index: 2.6.24-mm1/arch/x86/kernel/entry_64.S
=====
--- 2.6.24-mm1.orig/arch/x86/kernel/entry_64.S
+++ 2.6.24-mm1/arch/x86/kernel/entry_64.S
@@ -425,6 +425,7 @@ END(label)
    PTREGSCALL stub_rt_sigsuspend, sys_rt_sigsuspend, %rdx
    PTREGSCALL stub_sigaltstack, sys_sigaltstack, %rdx
    PTREGSCALL stub_iopl, sys_iopl, %rsi
+ PTREGSCALL stub_clone64, sys_clone64, %rcx

ENTRY(ptregscall_common)
    popq %r11
Index: 2.6.24-mm1/arch/powerpc/kernel/entry_32.S
=====
--- 2.6.24-mm1.orig/arch/powerpc/kernel/entry_32.S
+++ 2.6.24-mm1/arch/powerpc/kernel/entry_32.S
@@ -452,6 +452,14 @@ ppc_clone:
    stw r0,_TRAP(r1) /* register set saved */
    b sys_clone

+ .globl ppc_clone64
+ppc_clone64:
+ SAVE_NVGPRS(r1)
+ lwz r0,_TRAP(r1)
+ rlwinm r0,r0,0,0,30 /* clear LSB to indicate full */
+ stw r0,_TRAP(r1) /* register set saved */
+ b sys_clone64
+
+ .globl ppc_swapcontext
+ppc_swapcontext:
+ SAVE_NVGPRS(r1)
Index: 2.6.24-mm1/arch/powerpc/kernel/entry_64.S
=====
--- 2.6.24-mm1.orig/arch/powerpc/kernel/entry_64.S
+++ 2.6.24-mm1/arch/powerpc/kernel/entry_64.S
@@ -298,6 +298,11 @@ _GLOBAL(ppc_clone)
    bl .sys_clone
    b syscall_exit

+ _GLOBAL(ppc_clone64)
+ bl .save_nvgprs
+ bl .sys_clone64
+ b syscall_exit

```

```

+
_GLOBAL(ppc32_swapcontext)
    bl .save_nvgprs
    bl .compat_sys_swapcontext
Index: 2.6.24-mm1/arch/s390/kernel/compat_wrapper.S
=====
--- 2.6.24-mm1.orig/arch/s390/kernel/compat_wrapper.S
+++ 2.6.24-mm1/arch/s390/kernel/compat_wrapper.S
@@ -1712,3 +1712,9 @@ sys_fallocate_wrapper:
    slg %r5,%r6,32 # get high word of 64bit loff_t
    l %r5,164(%r15) # get low word of 64bit loff_t
    jg sys_fallocate
+
+ .globl sys_unshare64_wrapper
+sys_unshare64_wrapper:
+ llgr %r2,%r2 # unsigned long
+ llgr %r3,%r3 # unsigned long
+ jg sys_unshare64
Index: 2.6.24-mm1/kernel/fork.c
=====
--- 2.6.24-mm1.orig/kernel/fork.c
+++ 2.6.24-mm1/kernel/fork.c
@@ -1793,3 +1793,10 @@ asmlinkage long sys_unshare(unsigned lon
{
    return do_unshare(unshare_flags);
}
+
+asmlinkage long sys_unshare64(unsigned long flags_high, unsigned long flags_low)
+{
+ u64 unshare_flags = ((u64) flags_high << 32 | flags_low);
+
+ return do_unshare(unshare_flags);
+}
Index: 2.6.24-mm1/arch/x86/ia32/sys_ia32.c
=====
--- 2.6.24-mm1.orig/arch/x86/ia32/sys_ia32.c
+++ 2.6.24-mm1/arch/x86/ia32/sys_ia32.c
@@ -824,6 +824,18 @@ asmlinkage long sys32_clone(unsigned int
    return do_fork(clone_flags, newsp, regs, 0, parent_tid, child_tid);
}

+asmlinkage long sys32_clone64(unsigned int flags_high, unsigned int flags_low,
+ unsigned int newsp, struct pt_regs *regs)
+{
+ u64 clone_flags = ((u64) flags_high << 32 | flags_low);
+
+ clone_flags &= ~(CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID);
+

```

```

+ if (!newsp)
+ newsp = regs->sp;
+ return do_fork(clone_flags, newsp, regs, 0, NULL, NULL);
+}
+
+/*
+ * Some system calls that need sign extended arguments. This could be
+ * done by a generic wrapper.

```

Index: 2.6.24-mm1/arch/x86/ia32/ia32entry.S

```

=====
--- 2.6.24-mm1.orig/arch/x86/ia32/ia32entry.S
+++ 2.6.24-mm1/arch/x86/ia32/ia32entry.S
@@ -373,6 +373,7 @@ quiet_ni_syscall:
    PTREGSCALL stub32_vfork, sys_vfork, %rdi
    PTREGSCALL stub32_iopl, sys_iopl, %rsi
    PTREGSCALL stub32_rt_sigsuspend, sys_rt_sigsuspend, %rdx
+ PTREGSCALL stub32_clone64, sys32_clone64, %rcx

ENTRY(ia32_ptregs_common)
    popq %r11
@@ -727,4 +728,7 @@ ia32_sys_call_table:
    .quad sys32_fallocate
    .quad compat_sys_timerfd_settime /* 325 */
    .quad compat_sys_timerfd_gettime
+ .quad stub32_clone64
+ .quad sys_unshare64
+
ia32_syscall_end:

--

```

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

Subject: Re: [PATCH 3/3] add the clone64() and unshare64() syscalls
Posted by [Sukadev Bhattiprolu](#) on Thu, 10 Apr 2008 02:15:23 GMT
[View Forum Message](#) <> [Reply to Message](#)

Jakub Jelinek [jakub@redhat.com] wrote:
| On Wed, Apr 09, 2008 at 03:34:59PM -0700, sukadev@us.ibm.com wrote:
| > From: Cedric Le Goater <clg@fr.ibm.com>
| > Subject: [PATCH 3/3] add the clone64() and unshare64() syscalls
| >
| > This patch adds 2 new syscalls :
| >
| > long sys_clone64(unsigned long flags_high, unsigned long flags_low,

```
| > unsigned long newsp);
| >
| > long sys_unshare64(unsigned long flags_high, unsigned long flags_low);
|
| Can you explain why are you adding it for 64-bit arches too? unsigned long
| is there already 64-bit, and both sys_clone and sys_unshare have unsigned
| long flags, rather than unsigned int.
```

Hmm,

By simply resuing clone() on 64 bit and adding a new call for 32-bit won't the semantics of clone() differ between the two ?

i.e clone() on 64 bit supports say CLONE_NEWPTS clone() on 32bit does not ?

Wouldn't it be simpler/cleaner if clone() and clone64() behaved the same on both 32 and 64 bit systems ?

Sukadev

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

Subject: Re: [PATCH 3/3] add the clone64() and unshare64() syscalls
Posted by [hpa](#) on Thu, 10 Apr 2008 03:40:07 GMT
[View Forum Message](#) <> [Reply to Message](#)

sukadev@us.ibm.com wrote:

```
> Jakub Jelinek [jakub@redhat.com] wrote:
> | On Wed, Apr 09, 2008 at 03:34:59PM -0700, sukadev@us.ibm.com wrote:
> | > From: Cedric Le Goater <clg@fr.ibm.com>
> | > Subject: [PATCH 3/3] add the clone64() and unshare64() syscalls
> | >
> | > This patch adds 2 new syscalls :
> | >
> | > long sys_clone64(unsigned long flags_high, unsigned long flags_low,
> | > unsigned long newsp);
> | >
> | > long sys_unshare64(unsigned long flags_high, unsigned long flags_low);
> | >
> | Can you explain why are you adding it for 64-bit arches too? unsigned long
> | is there already 64-bit, and both sys_clone and sys_unshare have unsigned
> | long flags, rather than unsigned int.
>
> Hmm,
>
```

> By simply resuing clone() on 64 bit and adding a new call for 32-bit won't
> the semantics of clone() differ between the two ?
>
> i.e clone() on 64 bit supports say CLONE_NEWPTS clone() on 32bit does not ?
>
> Wouldn't it be simpler/cleaner if clone() and clone64() behaved the same
> on both 32 and 64 bit systems ?
>

No, not really. The way this work on the libc side is pretty much "use clone64 if it exists, otherwise use clone".

-hpa

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