

## [patch 06/12] x86: split FPU state from task state

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-02/msg10250.html>

---

- *From:* Ingo Molnar <mingo@xxxxxxx>
  - *Date:* Wed, 28 Feb 2007 22:41:57 +0100
- 

From: Arjan van de Ven <arjan@xxxxxxxxxxxxxxxx>

Split the FPU save area from the task struct. This allows easy migration of FPU context, and it's generally cleaner. It also allows the following two (future) optimizations:

- 1) allocate the right size for the actual cpu rather than 512 bytes always
- 2) only allocate when the application actually uses FPU, so in the first lazy FPU trap. This could save memory for non-fpu using apps.

Signed-off-by: Arjan van de Ven <arjan@xxxxxxxxxxxxxxxx>

Signed-off-by: Ingo Molnar <mingo@xxxxxxx>

---  
arch/i386/kernel/i387.c | 96 ++++++-----  
arch/i386/kernel/process.c | 56 ++++++  
arch/i386/kernel/traps.c | 10 ----  
include/asm-i386/i387.h | 6 +-  
include/asm-i386/processor.h | 6 ++  
include/asm-i386/thread\_info.h | 6 ++  
kernel/fork.c | 7 ++  
7 files changed, 123 insertions(+), 64 deletions(-)

Index: linux/arch/i386/kernel/i387.c

```
=====
--- linux.orig/arch/i386/kernel/i387.c
+++ linux/arch/i386/kernel/i387.c
@@ -31,9 +31,9 @@ void mxcsr_feature_mask_init(void)
unsigned long mask = 0;
clts();
if (cpu_has_fxr) {
- memset(&current->thread.i387.fxsave, 0, sizeof(struct i387_fxsave_struct));
- asm volatile("fxsave %0" : : "m" (current->thread.i387.fxsave));
- mask = current->thread.i387.fxsave.mxcsr_mask;
+ memset(&current->thread.i387->fxsave, 0, sizeof(struct i387_fxsave_struct));
+ asm volatile("fxsave %0" : : "m" (current->thread.i387->fxsave));
+ mask = current->thread.i387->fxsave.mxcsr_mask;
if (mask == 0) mask = 0x0000ffbf;
}
mxcsr_feature_mask &= mask;
```

[patch 06/12] x86: split FPU state from task state

```
@@ -49,16 +49,16 @@ void mxcsr_feature_mask_init(void)
void init_fpu(struct task_struct *tsk)
{
if (cpu_has_fxsr) {
- memset(&tsk->thread.i387.fxsave, 0, sizeof(struct i387_fxsave_struct));
- tsk->thread.i387.fxsave.cwd = 0x37f;
+ memset(&tsk->thread.i387->fxsave, 0, sizeof(struct i387_fxsave_struct));
+ tsk->thread.i387->fxsave.cwd = 0x37f;
if (cpu_has_xmm)
- tsk->thread.i387.fxsave.mxcsr = 0x1f80;
+ tsk->thread.i387->fxsave.mxcsr = 0x1f80;
} else {
- memset(&tsk->thread.i387.fssave, 0, sizeof(struct i387_fsave_struct));
- tsk->thread.i387.fssave.cwd = 0xffff037fu;
- tsk->thread.i387.fssave.swd = 0xffff0000u;
- tsk->thread.i387.fssave.twd = 0xfffffffffu;
- tsk->thread.i387.fssave.fos = 0xffff0000u;
+ memset(&tsk->thread.i387->fsave, 0, sizeof(struct i387_fsave_struct));
+ tsk->thread.i387->fsave.cwd = 0xffff037fu;
+ tsk->thread.i387->fsave.swd = 0xffff0000u;
+ tsk->thread.i387->fsave.twd = 0xfffffffffu;
+ tsk->thread.i387->fsave.fos = 0xffff0000u;
}
/* only the device not available exception or ptrace can call init_fpu */
set_stopped_child_used_math(tsk);
@@ -152,18 +152,18 @@ static inline unsigned long twd_fxsr_to_
unsigned short get_fpu_cwd( struct task_struct *tsk )
{
if ( cpu_has_fxsr ) {
- return tsk->thread.i387.fxsave.cwd;
+ return tsk->thread.i387->fxsave.cwd;
} else {
- return (unsigned short)tsk->thread.i387.fssave.cwd;
+ return (unsigned short)tsk->thread.i387->fsave.cwd;
}
}

unsigned short get_fpu_swd( struct task_struct *tsk )
{
if ( cpu_has_fxsr ) {
- return tsk->thread.i387.fxsave.swd;
+ return tsk->thread.i387->fxsave.swd;
} else {
- return (unsigned short)tsk->thread.i387.fssave.swd;
+ return (unsigned short)tsk->thread.i387->fsave.swd;
}
}

@@ -171,9 +171,9 @@ unsigned short get_fpu_swd( struct task_
unsigned short get_fpu_twd( struct task_struct *tsk )
{
```

[patch 06/12] x86: split FPU state from task state

```
if ( cpu_has_fxsr ) {
- return tsk->thread.i387.fxsave.twd;
+ return tsk->thread.i387->fxsave.twd;
} else {
- return (unsigned short)tsk->thread.i387.fsave.twd;
+ return (unsigned short)tsk->thread.i387->fsave.twd;
}
}
#endif /* 0 */
@@ -181,7 +181,7 @@ unsigned short get_fpu_twd( struct task_
unsigned short get_fpu_mxcsr( struct task_struct *tsk )
{
if ( cpu_has_xmm ) {
- return tsk->thread.i387.fxsave.mxcsr;
+ return tsk->thread.i387->fxsave.mxcsr;
} else {
return 0x1f80;
}
@@ -192,27 +192,27 @@ unsigned short get_fpu_mxcsr( struct tas
void set_fpu_cwd( struct task_struct *tsk, unsigned short cwd )
{
if ( cpu_has_fxsr ) {
- tsk->thread.i387.fxsave.cwd = cwd;
+ tsk->thread.i387->fxsave.cwd = cwd;
} else {
- tsk->thread.i387.fsave.cwd = ((long)cwd | 0xffff0000u);
+ tsk->thread.i387->fsave.cwd = ((long)cwd | 0xffff0000u);
}
}

void set_fpu_swd( struct task_struct *tsk, unsigned short swd )
{
if ( cpu_has_fxsr ) {
- tsk->thread.i387.fxsave.swd = swd;
+ tsk->thread.i387->fxsave.swd = swd;
} else {
- tsk->thread.i387.fsave.swd = ((long)swd | 0xffff0000u);
+ tsk->thread.i387->fsave.swd = ((long)swd | 0xffff0000u);
}
}

void set_fpu_twd( struct task_struct *tsk, unsigned short twd )
{
if ( cpu_has_fxsr ) {
- tsk->thread.i387.fxsave.twd = twd_i387_to_fxsr(twd);
+ tsk->thread.i387->fxsave.twd = twd_i387_to_fxsr(twd);
} else {
- tsk->thread.i387.fsave.twd = ((long)twd | 0xffff0000u);
+ tsk->thread.i387->fsave.twd = ((long)twd | 0xffff0000u);
}
}
```

[patch 06/12] x86: split FPU state from task state

```
@@ -298,8 +298,8 @@ static inline int save_i387_fsave( struct
struct task_struct *tsk = current;

unlazy_fpu( tsk );
- tsk->thread.i387.fsave.status = tsk->thread.i387.fsave.swd;
- if ( __copy_to_user( buf, &tsk->thread.i387.fsave,
+ tsk->thread.i387->fsave.status = tsk->thread.i387->fsave.swd;
+ if ( __copy_to_user( buf, &tsk->thread.i387->fsave,
sizeof(struct i387_fsave_struct) ) )
return -1;
return 1;
@@ -312,15 +312,15 @@ static int save_i387_fxsave( struct _fps

unlazy_fpu( tsk );

- if ( convert_fxsr_to_user( buf, &tsk->thread.i387.fxsave ) )
+ if ( convert_fxsr_to_user( buf, &tsk->thread.i387->fxsave ) )
return -1;

- err |= __put_user( tsk->thread.i387.fxsave.swd, &buf->status );
+ err |= __put_user( tsk->thread.i387->fxsave.swd, &buf->status );
err |= __put_user( X86_FXSR_MAGIC, &buf->magic );
if ( err )
return -1;

- if ( __copy_to_user( &buf->_fxsr_env[0], &tsk->thread.i387.fxsave,
+ if ( __copy_to_user( &buf->_fxsr_env[0], &tsk->thread.i387->fxsave,
sizeof(struct i387_fxsave_struct) ) )
return -1;
return 1;
@@ -343,7 +343,7 @@ int save_i387( struct _fpstate __user *b
return save_i387_fsave( buf );
}
} else {
- return save_i387_soft( &current->thread.i387.soft, buf );
+ return save_i387_soft( &current->thread.i387->soft, buf );
}
}

@@ -351,7 +351,7 @@ static inline int restore_i387_fsave( st
{
struct task_struct *tsk = current;
clear_fpu( tsk );
- return __copy_from_user( &tsk->thread.i387.fsave, buf,
+ return __copy_from_user( &tsk->thread.i387->fsave, buf,
sizeof(struct i387_fsave_struct) );
}

@@ -360,11 +360,11 @@ static int restore_i387_fxsave( struct _
int err;
```

[patch 06/12] x86: split FPU state from task state

```
struct task_struct *tsk = current;
clear_fpu( tsk );
- err = __copy_from_user( &tsk->thread.i387.fxsave, &buf->_fxsr_env[0],
+ err = __copy_from_user( &tsk->thread.i387->fxsave, &buf->_fxsr_env[0],
sizeof(struct i387_fxsave_struct) );
/* mxcsr reserved bits must be masked to zero for security reasons */
- tsk->thread.i387.fxsave.mxcsr &= mxcsr_feature_mask;
- return err ? 1 : convert_fxr_from_user( &tsk->thread.i387.fxsave, buf );
+ tsk->thread.i387->fxsave.mxcsr &= mxcsr_feature_mask;
+ return err ? 1 : convert_fxr_from_user( &tsk->thread.i387->fxsave, buf );
}
```

```
int restore_i387( struct _fpstate __user *buf )
@@ -378,7 +378,7 @@ int restore_i387( struct _fpstate __user
err = restore_i387_fsave( buf );
}
} else {
- err = restore_i387_soft( &current->thread.i387.soft, buf );
+ err = restore_i387_soft( &current->thread.i387->soft, buf );
}
set_used_math();
return err;
@@ -391,7 +391,7 @@ int restore_i387( struct _fpstate __user
static inline int get_fpregs_fsave( struct user_i387_struct __user *buf,
struct task_struct *tsk )
{
- return __copy_to_user( buf, &tsk->thread.i387.fssave,
+ return __copy_to_user( buf, &tsk->thread.i387->fsave,
sizeof(struct user_i387_struct) );
}
```

```
@@ -399,7 +399,7 @@ static inline int get_fpregs_fsave( str
struct task_struct *tsk )
{
return convert_fxr_to_user( (struct _fpstate __user *)buf,
- &tsk->thread.i387.fxsave );
+ &tsk->thread.i387->fxsave );
}
```

```
int get_fpregs( struct user_i387_struct __user *buf, struct task_struct *tsk )
@@ -411,7 +411,7 @@ int get_fpregs( struct user_i387_struct
return get_fpregs_fsave( buf, tsk );
}
} else {
- return save_i387_soft( &tsk->thread.i387.soft,
+ return save_i387_soft( &tsk->thread.i387->soft,
(struct _fpstate __user *)buf );
}
}
@@ -419,14 +419,14 @@ int get_fpregs( struct user_i387_struct
static inline int set_fpregs_fsave( struct task_struct *tsk,
```

[patch 06/12] x86: split FPU state from task state

```
struct user_i387_struct __user *buf )
{
- return __copy_from_user( &tsk->thread.i387.fsave, buf,
+ return __copy_from_user( &tsk->thread.i387->fsave, buf,
sizeof(struct user_i387_struct) );
}

static inline int set_fpregs_fxsave( struct task_struct *tsk,
struct user_i387_struct __user *buf )
{
- return convert_fxsr_from_user( &tsk->thread.i387.fxsave,
+ return convert_fxsr_from_user( &tsk->thread.i387->fxsave,
(struct _fpstate __user *)buf );
}

@@ -439,7 +439,7 @@ int set_fpregs( struct task_struct *tsk,
return set_fpregs_fsave( tsk, buf );
}
} else {
- return restore_i387_soft( &tsk->thread.i387.soft,
+ return restore_i387_soft( &tsk->thread.i387->soft,
(struct _fpstate __user *)buf );
}
}
@@ -447,7 +447,7 @@ int set_fpregs( struct task_struct *tsk,
int get_fpxregs( struct user_fxsr_struct __user *buf, struct task_struct *tsk )
{
if ( cpu_has_fxsr ) {
- if ( __copy_to_user( buf, &tsk->thread.i387.fxsave,
+ if ( __copy_to_user( buf, &tsk->thread.i387->fxsave,
sizeof(struct user_fxsr_struct) ))
return -EFAULT;
return 0;
@@ -461,11 +461,11 @@ int set_fpxregs( struct task_struct *tsk
int ret = 0;

if ( cpu_has_fxsr ) {
- if ( __copy_from_user( &tsk->thread.i387.fxsave, buf,
+ if ( __copy_from_user( &tsk->thread.i387->fxsave, buf,
sizeof(struct user_fxsr_struct) ))
ret = -EFAULT;
/* mxcsr reserved bits must be masked to zero for security reasons */
- tsk->thread.i387.fxsave.mxcsr &= mxcsr_feature_mask;
+ tsk->thread.i387->fxsave.mxcsr &= mxcsr_feature_mask;
} else {
ret = -EIO;
}
@@ -479,7 +479,7 @@ int set_fpxregs( struct task_struct *tsk
static inline void copy_fpu_fsave( struct task_struct *tsk,
struct user_i387_struct *fpu )
{
```

[patch 06/12] x86: split FPU state from task state

```
- memcpy( fpu, &tsk->thread.i387.fsave,
+ memcpy( fpu, &tsk->thread.i387->fsave,
sizeof(struct user_i387_struct) );
}

@@ -490,10 +490,10 @@ static inline void copy_fpu_fxsave( struct
unsigned short *from;
int i;

- memcpy( fpu, &tsk->thread.i387.fxsave, 7 * sizeof(long) );
+ memcpy( fpu, &tsk->thread.i387->fxsave, 7 * sizeof(long) );

to = (unsigned short *)&fpu->st_space[0];
- from = (unsigned short *)&tsk->thread.i387.fxsave.st_space[0];
+ from = (unsigned short *)&tsk->thread.i387->fxsave.st_space[0];
for ( i = 0 ; i < 8 ; i++, to += 5, from += 8 ) {
memcpy( to, from, 5 * sizeof(unsigned short) );
}
@@ -540,7 +540,7 @@ int dump_task_extended_fpu(struct task_s
if (fpvalid) {
if (tsk == current)
unlazy_fpu(tsk);
- memcpy(fpu, &tsk->thread.i387.fxsave, sizeof(*fpu));
+ memcpy(fpu, &tsk->thread.i387->fxsave, sizeof(*fpu));
}
return fpvalid;
}
Index: linux/arch/i386/kernel/process.c
```

```
-----
--- linux.orig/arch/i386/kernel/process.c
+++ linux/arch/i386/kernel/process.c
@@ -648,7 +648,7 @@ struct task_struct fastcall * __switch_t

/* we're going to use this soon, after a few expensive things */
if (next_p->fpu_counter > 5)
- prefetch(&next->i387.fxsave);
+ prefetch(&next->i387->fxsave);

/*
* Reload esp0.
@@ -927,3 +927,57 @@ unsigned long arch_align_stack(unsigned
sp -= get_random_int() % 8192;
return sp & ~0xf;
}
+
+
+
+struct kmem_cache *task_struct_cache;
+struct kmem_cache *task_i387_cache;
+
+struct task_struct * alloc_task_struct(void)
```

[patch 06/12] x86: split FPU state from task state

```
+{
+ struct task_struct *tsk;
+ tsk = kmem_cache_alloc(task_struct_cachep, GFP_KERNEL);
+ if (!tsk)
+ return NULL;
+ tsk->thread.i387 = kmem_cache_alloc(task_i387_cachep, GFP_KERNEL);
+ if (!tsk->thread.i387)
+ goto error;
+ WARN_ON((unsigned long)tsk->thread.i387 & 15);
+ return tsk;
+
+error:
+ kfree(tsk);
+ return NULL;
+}
+
+void memcpy_task_struct(struct task_struct *dst, struct task_struct *src)
+{
+ union i387_union *ptr;
+ ptr = dst->thread.i387;
+ *dst = *src;
+ dst->thread.i387 = ptr;
+ memcpy(dst->thread.i387, src->thread.i387, sizeof(union i387_union));
+}
+
+void free_task_struct(struct task_struct *tsk)
+{
+ kmem_cache_free(task_i387_cachep, tsk->thread.i387);
+ tsk->thread.i387=NULL;
+ kmem_cache_free(task_struct_cachep, tsk);
+}
+
+
+void task_struct_slab_init(void)
+{
+ /* create a slab on which task_structs can be allocated */
+ task_struct_cachep =
+ kmem_cache_create("task_struct", sizeof(struct task_struct),
+ ARCH_MIN_TASKALIGN, SLAB_PANIC, NULL, NULL);
+ task_i387_cachep =
+ kmem_cache_create("task_i387", sizeof(union i387_union), 32,
+ SLAB_PANIC | SLAB_MUST_HWCACHE_ALIGN, NULL, NULL);
+}
+
+
+/* the very init task needs a static allocated i387 area */
+union i387_union init_i387_context;
Index: linux/arch/i386/kernel/traps.c
=====
--- linux.orig/arch/i386/kernel/traps.c
+++ linux/arch/i386/kernel/traps.c
```

[patch 06/12] x86: split FPU state from task state

```
@@ -1157,16 +1157,6 @@ void __init trap_init(void)
set_trap_gate(19,&simd_coprocessor_error);
```

```
if (cpu_has_fxsr) {
- /*
- * Verify that the FXSAVE/FXRSTOR data will be 16-byte aligned.
- * Generates a compile-time "error: zero width for bit-field" if
- * the alignment is wrong.
- */
- struct fxsrAlignAssert {
- int _:(offsetof(struct task_struct,
- thread.i387.fxsave) & 15);
- };
-
printk(KERN_INFO "Enabling fast FPU save and restore... ");
set_in_cr4(X86_CR4_OSFXSR);
printk("done.\n");
Index: linux/include/asm-i386/i387.h
```

```
----- linux.orig/include/asm-i386/i387.h
+++ linux/include/asm-i386/i387.h
@@ -34,7 +34,7 @@ extern void init_fpu(struct task_struct
"nop ; frstor %1", \
"fxrstor %1", \
X86_FEATURE_FXSR, \
- "m" ((tsk)->thread.i387.fxsave))
+ "m" ((tsk)->thread.i387->fxsave))
```

```
extern void kernel_fpu_begin(void);
#define kernel_fpu_end() do { stts(); preempt_enable(); } while(0)
@@ -60,8 +60,8 @@ static inline void __save_init_fpu( stru
"fxsave %[fx]\n"
"bt $7,%[fsw] ; jnc 1f ; fnclex\n1:",
X86_FEATURE_FXSR,
- [fx] "m" (tsk->thread.i387.fxsave),
- [fsw] "m" (tsk->thread.i387.fxsave.swd) : "memory");
+ [fx] "m" (tsk->thread.i387->fxsave),
+ [fsw] "m" (tsk->thread.i387->fxsave.swd) : "memory");
/* AMD K7/K8 CPUs don't save/restore FDP/FIP/FOP unless an exception
is pending. Clear the x87 state here by setting it to fixed
values. safe_address is a random variable that should be in L1 */
Index: linux/include/asm-i386/processor.h
```

```
----- linux.orig/include/asm-i386/processor.h
+++ linux/include/asm-i386/processor.h
@@ -407,7 +407,7 @@ struct thread_struct {
/* fault info */
unsigned long cr2, trap_no, error_code;
/* floating point info */
- union i387_union i387;
+ union i387_union *i387;
```

[patch 06/12] x86: split FPU state from task state

```
/* virtual 86 mode info */
struct vm86_struct __user * vm86_info;
unsigned long screen_bitmap;
@@ -420,11 +420,15 @@ struct thread_struct {
unsigned long io_bitmap_max;
};

+
+extern union i387_union init_i387_context;
+
#define INIT_THREAD { \
.vm86_info = NULL, \
.sysenter_cs = __KERNEL_CS, \
.io_bitmap_ptr = NULL, \
.fs = __KERNEL_PDA, \
+ .i387 = &init_i387_context, \
}

/*
Index: linux/include/asm-i386/thread_info.h
=====
--- linux.orig/include/asm-i386/thread_info.h
+++ linux/include/asm-i386/thread_info.h
@@ -102,6 +102,12 @@ static inline struct thread_info *current

#define free_thread_info(info) kfree(info)

+#define __HAVE_ARCH_TASK_STRUCT_ALLOCATOR
+extern struct task_struct * alloc_task_struct(void);
+extern void free_task_struct(struct task_struct *tsk);
+extern void memcpy_task_struct(struct task_struct *dst, struct task_struct *src);
+extern void task_struct_slab_init(void);
+
#else /* !__ASSEMBLY__ */

/* how to get the thread information struct from ASM */
Index: linux/kernel/fork.c
=====
--- linux.orig/kernel/fork.c
+++ linux/kernel/fork.c
@@ -84,6 +84,8 @@ int nr_processes(void)
#ifdef __HAVE_ARCH_TASK_STRUCT_ALLOCATOR
# define alloc_task_struct() kmem_cache_alloc(task_struct_cachep, GFP_KERNEL)
# define free_task_struct(tsk) kmem_cache_free(task_struct_cachep, (tsk))
+# define memcpy_task_struct(dst, src) *dst = *src;
+
static struct kmem_cache *task_struct_cachep;
#endif

@@ -138,6 +140,8 @@ void __init fork_init(unsigned long memp
task_struct_cachep =
```

[patch 06/12] x86: split FPU state from task state

```
kmem_cache_create("task_struct", sizeof(struct task_struct),
ARCH_MIN_TASKALIGN, SLAB_PANIC, NULL, NULL);
+#else
+ task_struct_slab_init();
#endif

/*
@@ -176,7 +180,8 @@ static struct task_struct *dup_task_stru
return NULL;
}
```

```
- *tsk = *orig;
+ memcpy_task_struct(tsk, orig);
+
tsk->thread_info = ti;
setup_thread_stack(tsk, orig);
```

-  
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in  
the body of a message to majordomo@xxxxxxxxxxxxxxxxx  
More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
Please read the FAQ at <http://www.tux.org/lkml/>