

[PATCH] ppc64: Implement a vDSO and use it for signal trampoline

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-01/9183.html>

From: Benjamin Herrenschmidt (*benh_at_kernel.crashing.org*)

Date: 01/31/05

To: Andrew Morton <akpm@osdl.org>

Date: Mon, 31 Jan 2005 17:04:06 +1100

Hi !

This is a rather large patch. See notes below for possible backward compatibility issues. (Note: It depends on "ppc64: Move systemcfg out of head.S" beeing applied)

This patch adds to the ppc64 kernel a virtual .so (vDSO) that is mapped into every process space, similar to the x86 vsyscall page. However, the implementation is very different (and doesn't use the gate area mecanism). Actually, it contains two implementations, a 32 bits and a 64 bits one.

These vDSO's are currently mapped at 0x100000 (+1Mb) when possible (when a process load section isn't already there). In the future, we can randomize that address, or even imagine having a special phdr entry letting apps that wnat finer control over their address space to put it elsewhere (or not at all).

The implementation adds a hook to `binfmt_elf` to let the architecture add a real VMA to the process space instead of using the gate area mecanism. This mecanism wasn't very suitable for ppc, we couldn't just "shove" PTE entries mapping kernel addresses into userland without expensive changes to our hash table management. Instead, I made the vDSO be a normal VMA which, additionally, means it supports copy-on-write semantics if made writable via `ptrace/mprotect`, thus allowing breakpoints in the vDSO code.

The current implementation of the vDSOs contain the signal trampolines with appropriate DWARF informations, which enable us to use non-executable stacks (patches to come later) along with a few more functions that we hope glibc will soon make good use of (this is the "hard" part now :) Note that the symbols exposed by the vDSO aren't "normal" function symbols, apps can't be expected to link against them directly, the vDSO's are both seen as if they were linked at 0 and the symbols just contain offsets to the various functions. This is done on purpose to avoid a relocation step (ppc64 functions normally have descriptors with abs addresses in them). When glibc uses those functions, it's expected to use it's own trampolines that know how to reach them.

In some cases, the vDSO contains several versions of a given function (for various

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

CPUs), the kernel will "patch" the symbol table at boot to make it point to the appropriate one transparently.

What is currently implemented is:

```
– int __kernel_gettimeofday(struct timeval *tv, struct timezone *tz);
```

This is a fully userland implementation of gettimeofday, with no barriers and no locks, and providing 100% equivalent results to the syscall version

```
– void __kernel_sync_dicache(unsigned long start, unsigned long end)
```

This function sync's the data and instruction caches (for making data executable), it is expected that userland loaders use this instead of doing it themselves, as the kernel will provide optimized versions for the current CPU. Currently, the vDSO provides a full one for all CPUs prior to POWER5 and a nop one for POWER5 which implements hardware snooping at the L1 level. In the future, an intermediate implementation may be done for the POWER4 and 970 which don't need the "dcbst" loop (the L1D cache is write-through on those).

```
– void *__kernel_get_syscall_map(unsigned int *syscall_count) ;
```

Returns a pointer to a map of implemented syscalls on the currently running kernel. The map is agnostic to the size of "long", unlike kernel bitops, it stores bits from top to bottom so that memory actually contains a linear bitmap check for syscall N by testing bit $(0x80000000 \gg (N \& 0x1f))$ of $* 32$ bits int at $N \gg 5$.

Note about backward compatibility issues: A bug in the ppc64 libgcc unwinder makes it unable to unwind stacks properly across signals if the signal trampoline isn't on the stack. This has been fixed in CVS for gcc 4.0 and will be soon on the stable branch, but the problem exist will all currently used versions.

That means that until glibc gets the patch to enable it's use of the vDSO symbols for the DWARF unwinder (rather trivial patch that will be pushed to glibc CVS soon hopefully), unwinding from a signal handler will not work for 64 bits applications.

I consider this as a non–issue though as a patch is about to be produced, which can easily get pushed to "live" distros like debian, gentoo, fedora, etc... soon enough (it breaks compatibility with kernels below 2.4.20 unfortunately as our signal stack layout changed, crap crap crap), as there are few 64 bits applications out there (expect gentoo), as it's only really an issue with C++ code relying on throwing exceptions out of signal handlers (extremely rare it seems), and as "release" distros like SLES or RHEL will probably have the vDSO enabled glibc _and_ the unwinder fix by the time they release a version with a 2.6.11 or 2.6.12 kernel anyway :)

So far, I yet have to see an app failing because of that...

Finally, many many many thanks to Alan Modra for writing the DWARF information of the signal handlers and debugging the libgcc issues !

Signed–off–by: Benjamin Herrenschmidt <benh@kernel.crashing.org>

[PATCH] ppc64: Implement a vDSO and use it for signal trampoline

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

Index: linux-work/arch/ppc64/Makefile

```
----- linux-work.orig/arch/ppc64/Makefile 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/Makefile 2005-01-31 16:25:55.000000000 +1100
@@ -53,6 +53,8 @@
```

```
libs-y += arch/ppc64/lib/
core-y += arch/ppc64/kernel/
+core-y += arch/ppc64/kernel/vdso32/
+core-y += arch/ppc64/kernel/vdso64/
core-y += arch/ppc64/mm/
core-$(CONFIG_XMON) += arch/ppc64/xmon/
drivers-$(CONFIG_OPROFILE) += arch/ppc64/oprofile/
```

Index: linux-work/arch/ppc64/kernel/asm-offsets.c

```
----- linux-work.orig/arch/ppc64/kernel/asm-offsets.c 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/kernel/asm-offsets.c 2005-01-31 16:25:56.000000000 +1100
@@ -22,6 +22,7 @@
```

```
#include <linux/types.h>
#include <linux/mman.h>
#include <linux/mm.h>
+#include <linux/time.h>
#include <linux/hardirq.h>
#include <asm/io.h>
#include <asm/page.h>
@@ -35,6 +36,8 @@
#include <asm/rtas.h>
#include <asm/cputable.h>
#include <asm/cache.h>
+#include <asm/systemcfg.h>
+#include <asm/compat.h>
```

```
#define DEFINE(sym, val) \
asm volatile("\n->" #sym " %0 " #val : : "i" (val))
@@ -167,5 +170,24 @@
DEFINE(CPU_SPEC_FEATURES, offsetof(struct cpu_spec, cpu_features));
DEFINE(CPU_SPEC_SETUP, offsetof(struct cpu_spec, cpu_setup));
```

```
+ /* systemcfg offsets for use by vdso */
+ DEFINE(CFG_TB_ORIG_STAMP, offsetof(struct systemcfg, tb_orig_stamp));
+ DEFINE(CFG_TB_TICKS_PER_SEC, offsetof(struct systemcfg, tb_ticks_per_sec));
+ DEFINE(CFG_TB_TO_XS, offsetof(struct systemcfg, tb_to_xs));
+ DEFINE(CFG_STAMP_XSEC, offsetof(struct systemcfg, stamp_xsec));
+ DEFINE(CFG_TB_UPDATE_COUNT, offsetof(struct systemcfg, tb_update_count));
+ DEFINE(CFG_TZ_MINUTEWEST, offsetof(struct systemcfg, tz_minuteswest));
+ DEFINE(CFG_TZ_DSTTIME, offsetof(struct systemcfg, tz_dsttime));
+ DEFINE(CFG_SYSCALL_MAP32, offsetof(struct systemcfg, syscall_map_32));
+ DEFINE(CFG_SYSCALL_MAP64, offsetof(struct systemcfg, syscall_map_64));
+
+ /* timeval/timzone offsets for use by vdso */
+ DEFINE(TVAL64_TV_SEC, offsetof(struct timeval, tv_sec));
```

[PATCH] ppc64: Implement a vDSO and use it for signal trampoline

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ DEFINE(TVAL64_TV_USEC, offsetof(struct timeval, tv_usec));
+ DEFINE(TVAL32_TV_SEC, offsetof(struct compat_timeval, tv_sec));
+ DEFINE(TVAL32_TV_USEC, offsetof(struct compat_timeval, tv_usec));
+ DEFINE(TZONE_TZ_MINWEST, offsetof(struct timezone, tz_minuteswest));
+ DEFINE(TZONE_TZ_DSTTIME, offsetof(struct timezone, tz_dsttime));
+
+     return 0;
+ }
}
```

Index: linux-work/arch/ppc64/kernel/Makefile

```
----- linux-work.orig/arch/ppc64/kernel/Makefile 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/kernel/Makefile 2005-01-31 16:25:56.000000000 +1100
@@ -11,7 +11,7 @@
         udbg.o binfmt_elf32.o sys_ppc32.o ioctl32.o \
         ptrace32.o signal32.o rtc.o init_task.o \
         lmb.o cputable.o cpu_setup_power4.o idle_power4.o \
- iommu.o sysfs.o
+ iommu.o sysfs.o vdso.o

obj-$(CONFIG_PPC_OF) += of_device.o
```

Index: linux-work/arch/ppc64/kernel/signal32.c

```
----- linux-work.orig/arch/ppc64/kernel/signal32.c 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/kernel/signal32.c 2005-01-31 16:25:56.000000000 +1100
@@ -31,6 +31,7 @@
#include <asm/ppcdebug.h>
#include <asm/unistd.h>
#include <asm/cacheflush.h>
+#include <asm/vdso.h>

#define DEBUG_SIG 0

@@ -656,18 +657,24 @@

        /* Save user registers on the stack */
        frame = &rt_sf->uc.uc_mcontext;
- if (save_user_regs(regs, frame, __NR_rt_sigreturn))
- goto badframe;
-
+ if (put_user(regs->gpr[1], (unsigned long __user *)newsp))
+ goto badframe;
+
+ if (vdso32_rt_sigtramp && current->thread.vdso_base) {
+ if (save_user_regs(regs, frame, 0))
+ goto badframe;
+ regs->link = current->thread.vdso_base + vdso32_rt_sigtramp;
+ } else {
+ if (save_user_regs(regs, frame, __NR_rt_sigreturn))
+ goto badframe;
+ regs->link = (unsigned long) frame->tramp;
}
```

[PATCH] ppc64: Implement a vDSO and use it for signal trampoline

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ }
    regs->gpr[1] = (unsigned long) newsp;
    regs->gpr[3] = sig;
    regs->gpr[4] = (unsigned long) &rt_sf->info;
    regs->gpr[5] = (unsigned long) &rt_sf->uc;
    regs->gpr[6] = (unsigned long) rt_sf;
    regs->nip = (unsigned long) ka->sa.sa_handler;
- regs->link = (unsigned long) frame->tramp;
    regs->trap = 0;
    regs->result = 0;

@@ -825,8 +832,15 @@
    || __put_user(sig, &sc->signal))
        goto badframe;

- if (save_user_regs(regs, &frame->mctx, __NR_sigreturn))
- goto badframe;
+ if (vdso32_sigtramp && current->thread.vdso_base) {
+ if (save_user_regs(regs, &frame->mctx, 0))
+ goto badframe;
+ regs->link = current->thread.vdso_base + vdso32_sigtramp;
+ } else {
+ if (save_user_regs(regs, &frame->mctx, __NR_sigreturn))
+ goto badframe;
+ regs->link = (unsigned long) frame->mctx.tramp;
+ }

        if (put_user(regs->gpr[1], (unsigned long __user *)newsp))
            goto badframe;
@@ -834,7 +848,6 @@
    regs->gpr[3] = sig;
    regs->gpr[4] = (unsigned long) sc;
    regs->nip = (unsigned long) ka->sa.sa_handler;
- regs->link = (unsigned long) frame->mctx.tramp;
    regs->trap = 0;
    regs->result = 0;
```

Index: linux-work/arch/ppc64/kernel/setup.c

```
=====
--- linux-work.orig/arch/ppc64/kernel/setup.c 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/kernel/setup.c 2005-01-31 16:25:56.000000000 +1100
@@ -990,6 +990,34 @@
 }

/*
+ * Called from setup_arch to initialize the bitmap of available
+ * syscalls in the systemcfg page
+ */
+void __init setup_syscall_map(void)
+{
+ unsigned int i, count64 = 0, count32 = 0;
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ extern unsigned long *sys_call_table;
+ extern unsigned long *sys_call_table32;
+ extern unsigned long sys_ni_syscall;
+
+
+ for (i = 0; i < __NR_syscalls; i++) {
+ if (sys_call_table[i] == sys_ni_syscall)
+ continue;
+ count64++;
+ systemcfg->syscall_map_64[i >> 5] |= 0x80000000UL >> (i & 0x1f);
+ }
+ for (i = 0; i < __NR_syscalls; i++) {
+ if (sys_call_table32[i] == sys_ni_syscall)
+ continue;
+ count32++;
+ systemcfg->syscall_map_32[i >> 5] |= 0x80000000UL >> (i & 0x1f);
+ }
+ printk(KERN_INFO "Syscall map setup, %d 32 bits and %d 64 bits syscalls\n",
+ count32, count64);
+ }
+
+ /*
+  * Called into from start_kernel, after lock_kernel has been called.
+  * Initializes bootmem, which is used to manage page allocation until
+  * mem_init is called.
+  @@ -1027,6 +1055,9 @@
+     /* set up the bootmem stuff with available memory */
+     do_init_bootmem();

+ /* initialize the syscall map in systemcfg */
+ setup_syscall_map();
+
+     ppc_md.setup_arch();

+     /* Select the correct idle loop for the platform. */
Index: linux-work/arch/ppc64/kernel/signal.c
=====
--- linux-work.orig/arch/ppc64/kernel/signal.c 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/kernel/signal.c 2005-01-31 16:25:56.000000000 +1100
@@ -34,6 +34,7 @@
#include <asm/ppcdebug.h>
#include <asm/unistd.h>
#include <asm/cacheflush.h>
+#include <asm/vdso.h>

#define DEBUG_SIG 0

@@ -426,10 +427,14 @@
    goto badframe;

    /* Set up to return from userspace. */
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
- err |= setup_trampoline(__NR_rt_sigreturn, &frame->tramp[0]);
- if (err)
- goto badframe;
-
+ if (vdso64_rt_sigtramp && current->thread.vdso_base) {
+ regs->link = current->thread.vdso_base + vdso64_rt_sigtramp;
+ } else {
+ err |= setup_trampoline(__NR_rt_sigreturn, &frame->tramp[0]);
+ if (err)
+ goto badframe;
+ regs->link = (unsigned long) &frame->tramp[0];
+ }
    funct_desc_ptr = (func_descr_t __user *) ka->sa.sa_handler;

    /* Allocate a dummy caller frame for the signal handler. */
@@ -438,7 +443,6 @@

    /* Set up "regs" so we "return" to the signal handler. */
    err |= get_user(regs->nip, &funct_desc_ptr->entry);
- regs->link = (unsigned long) &frame->tramp[0];
    regs->gpr[1] = newsp;
    err |= get_user(regs->gpr[2], &funct_desc_ptr->toc);
    regs->gpr[3] = signr;
```

Index: linux-work/arch/ppc64/kernel/smp.c

```
=====
--- linux-work.orig/arch/ppc64/kernel/smp.c 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/kernel/smp.c 2005-01-31 16:25:56.000000000 +1100
@@ -383,7 +383,7 @@
    * For now we leave it which means the time can be some
    * number of msecs off until someone does a settimeofday()
    */
- do_gtod.tb_orig_stamp = tb_last_stamp;
+ do_gtod.varp->tb_orig_stamp = tb_last_stamp;
    systemcfg->tb_orig_stamp = tb_last_stamp;
#endif
```

Index: linux-work/arch/ppc64/kernel/time.c

```
=====
--- linux-work.orig/arch/ppc64/kernel/time.c 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/kernel/time.c 2005-01-31 16:25:56.000000000 +1100
@@ -86,8 +86,6 @@
unsigned long tb_ticks_per_jiffy;
unsigned long tb_ticks_per_usec = 100; /* sane default */
unsigned long tb_ticks_per_sec;
-unsigned long next_xtime_sync_tb;
-unsigned long xtime_sync_interval;
unsigned long tb_to_xs;
unsigned long tb_to_us;
unsigned long processor_freq;
@@ -158,8 +156,8 @@
    * The conversion to microseconds at the end is done
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
* without a divide (and in fact, without a multiply)
*/
- tb_ticks = tb_val - do_gtod.tb_orig_stamp;
  temp_varp = do_gtod.varp;
+ tb_ticks = tb_val - temp_varp->tb_orig_stamp;
  temp_tb_to_xs = temp_varp->tb_to_xs;
  temp_stamp_xsec = temp_varp->stamp_xsec;
  tb_xsec = mulhdu( tb_ticks, temp_tb_to_xs );
@@ -185,17 +183,55 @@
{
    struct timeval my_tv;

- if (cur_tb > next_xtime_sync_tb) {
- next_xtime_sync_tb = cur_tb + xtime_sync_interval;
- __do_gettimeofday(&my_tv, cur_tb);
-
- if (xtime.tv_sec <= my_tv.tv_sec) {
- xtime.tv_sec = my_tv.tv_sec;
- xtime.tv_nsec = my_tv.tv_usec * 1000;
- }
+ __do_gettimeofday(&my_tv, cur_tb);
+
+ if (xtime.tv_sec <= my_tv.tv_sec) {
+ xtime.tv_sec = my_tv.tv_sec;
+ xtime.tv_nsec = my_tv.tv_usec * 1000;
+ }
}

+/*
+ * When the timebase - tb_orig_stamp gets too big, we do a manipulation
+ * between tb_orig_stamp and stamp_xsec. The goal here is to keep the
+ * difference tb - tb_orig_stamp small enough to always fit inside a
+ * 32 bits number. This is a requirement of our fast 32 bits userland
+ * implementation in the vdso. If we "miss" a call to this function
+ * (interrupt latency, CPU locked in a spinlock, ...) and we end up
+ * with a too big difference, then the vdso will fallback to calling
+ * the syscall
+ */
+static __inline__ void timer_recalc_offset(unsigned long cur_tb)
+{
+ struct gettimeofday_vars * temp_varp;
+ unsigned temp_idx;
+ unsigned long offset, new_stamp_xsec, new_tb_orig_stamp;
+
+ if (((cur_tb - do_gtod.varp->tb_orig_stamp) & 0x80000000u) == 0)
+ return;
+
+ temp_idx = (do_gtod.var_idx == 0);
+ temp_varp = &do_gtod.vars[temp_idx];
+
+ new_tb_orig_stamp = cur_tb;
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```

+ offset = new_tb_orig_stamp - do_gtod.varp->tb_orig_stamp;
+ new_stamp_xsec = do_gtod.varp->stamp_xsec + mulhdu(offset, do_gtod.varp->tb_to_xs);
+
+ temp_varp->tb_to_xs = do_gtod.varp->tb_to_xs;
+ temp_varp->tb_orig_stamp = new_tb_orig_stamp;
+ temp_varp->stamp_xsec = new_stamp_xsec;
+ mb();
+ do_gtod.varp = temp_varp;
+ do_gtod.var_idx = temp_idx;
+
+ ++(systemcfg->tb_update_count);
+ wmb();
+ systemcfg->tb_orig_stamp = new_tb_orig_stamp;
+ systemcfg->stamp_xsec = new_stamp_xsec;
+ wmb();
+ ++(systemcfg->tb_update_count);
+}
+
#ifdef CONFIG_SMP
unsigned long profile_pc(struct pt_regs *regs)
{
@@ -311,6 +347,7 @@
    if (cpu == boot_cpuid) {
        write_seqlock(&xtime_lock);
        tb_last_stamp = lpaca->next_jiffy_update_tb;
+ timer_recalc_offset(lpaca->next_jiffy_update_tb);
        do_timer(regs);
        timer_sync_xtime(lpaca->next_jiffy_update_tb);
        timer_check_rtc();
@@ -398,7 +435,9 @@
    time_maxerror = NTP_PHASE_LIMIT;
    time_esterror = NTP_PHASE_LIMIT;

- delta_xsec = mulhdu( (tb_last_stamp-do_gtod.tb_orig_stamp), do_gtod.varp->tb_to_xs );
+ delta_xsec = mulhdu( (tb_last_stamp-do_gtod.varp->tb_orig_stamp),
+ do_gtod.varp->tb_to_xs );
+
    new_xsec = (new_nsec * XSEC_PER_SEC) / NSEC_PER_SEC;
    new_xsec += new_sec * XSEC_PER_SEC;
    if ( new_xsec > delta_xsec ) {
@@ -411,7 +450,7 @@
        * before 1970 ... eg. we booted ten days ago, and we are setting
        * the time to Jan 5, 1970 */
        do_gtod.varp->stamp_xsec = new_xsec;
- do_gtod.tb_orig_stamp = tb_last_stamp;
+ do_gtod.varp->tb_orig_stamp = tb_last_stamp;
        systemcfg->stamp_xsec = new_xsec;
        systemcfg->tb_orig_stamp = tb_last_stamp;
    }
@@ -464,9 +503,9 @@
    xtime.tv_sec = mktime(tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,

```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
        tm.tm_hour, tm.tm_min, tm.tm_sec);
    tb_last_stamp = get_tb();
- do_gtod.tb_orig_stamp = tb_last_stamp;
    do_gtod.varp = &do_gtod.vars[0];
    do_gtod.var_idx = 0;
+ do_gtod.varp->tb_orig_stamp = tb_last_stamp;
    do_gtod.varp->stamp_xsec = xtime.tv_sec * XSEC_PER_SEC;
    do_gtod.tb_ticks_per_sec = tb_ticks_per_sec;
    do_gtod.varp->tb_to_xs = tb_to_xs;
@@ -477,9 +516,6 @@
    systemcfg->stamp_xsec = xtime.tv_sec * XSEC_PER_SEC;
    systemcfg->tb_to_xs = tb_to_xs;

- xtime_sync_interval = tb_ticks_per_sec - (tb_ticks_per_sec/8);
- next_xtime_sync_tb = tb_last_stamp + xtime_sync_interval;
-
    time_freq = 0;

    xtime.tv_nsec = 0;
@@ -584,12 +620,12 @@
        stamp_xsec which is the time (in 1/2^20 second units) corresponding to tb_orig_stamp. This
        new value of stamp_xsec compensates for the change in frequency (implied by the new tb_to_xs)
        which guarantees that the current time remains the same */
- tb_ticks = get_tb() - do_gtod.tb_orig_stamp;
+ write_seqlock_irqsave( &xtime_lock, flags );
+ tb_ticks = get_tb() - do_gtod.varp->tb_orig_stamp;
    div128_by_32( 1024*1024, 0, new_tb_ticks_per_sec, &divres );
    new_tb_to_xs = divres.result_low;
    new_xsec = mulhdu( tb_ticks, new_tb_to_xs );

- write_seqlock_irqsave( &xtime_lock, flags );
    old_xsec = mulhdu( tb_ticks, do_gtod.varp->tb_to_xs );
    new_stamp_xsec = do_gtod.varp->stamp_xsec + old_xsec - new_xsec;

@@ -597,16 +633,12 @@
    values in do_gettimeofday. We alternate the copies and as long as a reasonable time elapses between
    changes, there will never be inconsistent values. ntpd has a minimum of one minute between updates
*/

- if (do_gtod.var_idx == 0) {
- temp_varp = &do_gtod.vars[1];
- temp_idx = 1;
- }
- else {
- temp_varp = &do_gtod.vars[0];
- temp_idx = 0;
- }
+ temp_idx = (do_gtod.var_idx == 0);
+ temp_varp = &do_gtod.vars[temp_idx];
+
    temp_varp->tb_to_xs = new_tb_to_xs;
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
temp_varp->stamp_xsec = new_stamp_xsec;
+ temp_varp->tb_orig_stamp = do_gtod.varp->tb_orig_stamp;
  mb();
  do_gtod.varp = temp_varp;
  do_gtod.var_idx = temp_idx;
```

Index: linux-work/arch/ppc64/kernel/vdso.c

```
=====
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-work/arch/ppc64/kernel/vdso.c 2005-01-31 16:25:56.000000000 +1100
@@ -0,0 +1,614 @@
+/*
+ * linux/arch/ppc64/kernel/vdso.c
+ *
+ * Copyright (C) 2004 Benjamin Herrenschmidt, IBM Corp.
+ * <benh@kernel.crashing.org>
+ *
+ * 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; either version
+ * 2 of the License, or (at your option) any later version.
+ */
+
+#include <linux/config.h>
+#include <linux/module.h>
+#include <linux/errno.h>
+#include <linux/sched.h>
+#include <linux/kernel.h>
+#include <linux/mm.h>
+#include <linux/smp.h>
+#include <linux/smp_lock.h>
+#include <linux/stddef.h>
+#include <linux/unistd.h>
+#include <linux/slab.h>
+#include <linux/user.h>
+#include <linux/elf.h>
+#include <linux/security.h>
+#include <linux/bootmem.h>
+
+#include <asm/pgtable.h>
+#include <asm/system.h>
+#include <asm/processor.h>
+#include <asm/mmu.h>
+#include <asm/mmu_context.h>
+#include <asm/machdep.h>
+#include <asm/cputable.h>
+#include <asm/sections.h>
+#include <asm/vdso.h>
+
+#undef DEBUG
+
+#ifdef DEBUG
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ #define DBG(fmt...) printk(fmt)
+ #else
+ #define DBG(fmt...)
+ #endif
+
+
+ /*
+  * The vDSOs themselves are here
+  */
+ extern char vdso64_start, vdso64_end;
+ extern char vdso32_start, vdso32_end;
+
+
+ static void *vdso64_kbase = &vdso64_start;
+ static void *vdso32_kbase = &vdso32_start;
+
+
+ unsigned int vdso64_pages;
+ unsigned int vdso32_pages;
+
+
+ /* Signal trampolines user addresses */
+
+
+ unsigned long vdso64_rt_sigtramp;
+ unsigned long vdso32_sigtramp;
+ unsigned long vdso32_rt_sigtramp;
+
+
+ /* Format of the patch table */
+ struct vdso_patch_def
+ {
+     u32 pvr_mask, pvr_value;
+     const char *gen_name;
+     const char *fix_name;
+ };
+
+
+ /* Table of functions to patch based on the CPU type/revision
+  *
+  * TODO: Improve by adding whole lists for each entry
+  */
+ static struct vdso_patch_def vdso_patches[] = {
+     {
+         0xffff0000, 0x003a0000, /* POWER5 */
+         "__kernel_sync_dicache", "__kernel_sync_dicache_p5"
+     },
+     {
+         0xffff0000, 0x003b0000, /* POWER5 */
+         "__kernel_sync_dicache", "__kernel_sync_dicache_p5"
+     },
+ };
+
+
+ /*
+  * Some infos carried around for each of them during parsing at
+  * boot time.
+  */
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+struct lib32_elfinfo
+{
+ Elf32_Ehdr *hdr; /* ptr to ELF */
+ Elf32_Sym *dynsym; /* ptr to .dynsym section */
+ unsigned long dynsymsize; /* size of .dynsym section */
+ char *dynstr; /* ptr to .dynstr section */
+ unsigned long text; /* offset of .text section in .so */
+};
+
+struct lib64_elfinfo
+{
+ Elf64_Ehdr *hdr;
+ Elf64_Sym *dynsym;
+ unsigned long dynsymsize;
+ char *dynstr;
+ unsigned long text;
+};
+
+#ifdef __DEBUG
+static void dump_one_vdso_page(struct page *pg, struct page *upg)
+{
+ printk("kpg: %p (c:%d,f:%08lx)", __va(page_to_pfn(pg) << PAGE_SHIFT),
+ page_count(pg),
+ pg->flags);
+ if (upg/* && pg != upg*/) {
+ printk(" upg: %p (c:%d,f:%08lx)", __va(page_to_pfn(upg) << PAGE_SHIFT),
+ page_count(upg),
+ upg->flags);
+ }
+ printk("\n");
+}
+
+static void dump_vdso_pages(struct vm_area_struct * vma)
+{
+ int i;
+
+ if (!vma || test_thread_flag(TIF_32BIT)) {
+ printk("vDSO32 @ %016lx:\n", (unsigned long)vds032_kbase);
+ for (i=0; i<vds032_pages; i++) {
+ struct page *pg = virt_to_page(vds032_kbase + i*PAGE_SIZE);
+ struct page *upg = (vma && vma->vm_mm) ?
+ follow_page(vma->vm_mm, vma->vm_start + i*PAGE_SIZE, 0)
+ : NULL;
+ dump_one_vdso_page(pg, upg);
+ }
+ }
+ if (!vma || !test_thread_flag(TIF_32BIT)) {
+ printk("vDSO64 @ %016lx:\n", (unsigned long)vds064_kbase);
+ for (i=0; i<vds064_pages; i++) {
+ struct page *pg = virt_to_page(vds064_kbase + i*PAGE_SIZE);
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ struct page *upg = (vma && vma->vm_mm) ?
+ follow_page(vma->vm_mm, vma->vm_start + i*PAGE_SIZE, 0)
+ : NULL;
+ dump_one_vdso_page(pg, upg);
+ }
+ }
+}
+ #endif /* DEBUG */
+
+ /*
+  * Keep a dummy vma_close for now, it will prevent VMA merging.
+  */
+ static void vdso_vma_close(struct vm_area_struct * vma)
+ {
+ }
+
+ /*
+  * Our nopage() function, maps in the actual vDSO kernel pages, they will
+  * be mapped read-only by do_no_page(), and eventually COW'ed, either
+  * right away for an initial write access, or by do_wp_page().
+  */
+ static struct page * vdso_vma_nopage(struct vm_area_struct * vma,
+ unsigned long address, int *type)
+ {
+ unsigned long offset = address - vma->vm_start;
+ struct page *pg;
+ void *vbase = test_thread_flag(TIF_32BIT) ? vdso32_kbase : vdso64_kbase;
+
+ DBG("vdso_vma_nopage(current: %s, address: %016lx, off: %lx)\n",
+ current->comm, address, offset);
+
+ if (address < vma->vm_start || address > vma->vm_end)
+ return NOPAGE_SIGBUS;
+
+ /*
+  * Last page is systemcfg, special handling here, no get_page() a
+  * this is a reserved page
+  */
+ if ((vma->vm_end - address) <= PAGE_SIZE)
+ return virt_to_page(systemcfg);
+
+ pg = virt_to_page(vbase + offset);
+ get_page(pg);
+ DBG(" ->page count: %d\n", page_count(pg));
+
+ return pg;
+ }
+
+ static struct vm_operations_struct vdso_vmops = {
+ .close = vdso_vma_close,
+ .nopage = vdso_vma_nopage,
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+};  
+  
+/*  
+ * This is called from binfmt_elf, we create the special vma for the  
+ * vDSO and insert it into the mm struct tree  
+ */  
+int arch_setup_additional_pages(struct linux_binprm *bprm, int executable_stack)  
+{  
+ struct mm_struct *mm = current->mm;  
+ struct vm_area_struct *vma;  
+ unsigned long vdso_pages;  
+ unsigned long vdso_base;  
+  
+ if (test_thread_flag(TIF_32BIT)) {  
+ vdso_pages = vdso32_pages;  
+ vdso_base = VDSO32_MBASE;  
+ } else {  
+ vdso_pages = vdso64_pages;  
+ vdso_base = VDSO64_MBASE;  
+ }  
+  
+ /* vDSO has a problem and was disabled, just don't "enable" it for the  
+ * process  
+ */  
+ if (vdso_pages == 0) {  
+ current->thread.vdso_base = 0;  
+ return 0;  
+ }  
+ vma = kmem_cache_alloc(vm_area_cachep, SLAB_KERNEL);  
+ if (vma == NULL)  
+ return -ENOMEM;  
+ if (security_vm_enough_memory(vdso_pages)) {  
+ kmem_cache_free(vm_area_cachep, vma);  
+ return -ENOMEM;  
+ }  
+ memset(vma, 0, sizeof(*vma));  
+  
+ /*  
+ * pick a base address for the vDSO in process space. We have a default  
+ * base of 1Mb on which we had a random offset up to 1Mb.  
+ * XXX: Add possibility for a program header to specify that location  
+ */  
+ current->thread.vdso_base = vdso_base;  
+ /* + ((unsigned long)vma & 0x000ff000); */  
+  
+ vma->vm_mm = mm;  
+ vma->vm_start = current->thread.vdso_base;  
+  
+ /*  
+ * the VMA size is one page more than the vDSO since systemcfg  
+ * is mapped in the last one
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ */
+ vma->vm_end = vma->vm_start + ((vdso_pages + 1) << PAGE_SHIFT);
+
+ /*
+ * our vma flags don't have VM_WRITE so by default, the process isn't allowed
+ * to write those pages.
+ * gdb can break that with ptrace interface, and thus trigger COW on those
+ * pages but it's then your responsibility to never do that on the "data" page
+ * of the vDSO or you'll stop getting kernel updates and your nice userland
+ * gettimeofday will be totally dead. It's fine to use that for setting
+ * breakpoints in the vDSO code pages though
+ */
+ vma->vm_flags = VM_READ | VM_EXEC | VM_MAYREAD | VM_MAYWRITE | VM_MAYEXEC;
+ vma->vm_flags |= mm->def_flags;
+ vma->vm_page_prot = protection_map[vma->vm_flags & 0x7];
+ vma->vm_ops = &vdso_vmops;
+
+ down_write(&mm->mmap_sem);
+ insert_vm_struct(mm, vma);
+ mm->total_vm += (vma->vm_end - vma->vm_start) >> PAGE_SHIFT;
+ up_write(&mm->mmap_sem);
+
+ return 0;
+}
+
+static void * __init find_section32(Elf32_Ehdr *ehdr, const char *secname,
+ unsigned long *size)
+{
+ Elf32_Shdr *sechdrs;
+ unsigned int i;
+ char *secnames;
+
+ /* Grab section headers and strings so we can tell who is who */
+ sechdrs = (void *)ehdr + ehdr->e_shoff;
+ secnames = (void *)ehdr + sechdrs[ehdr->e_shstrndx].sh_offset;
+
+ /* Find the section they want */
+ for (i = 1; i < ehdr->e_shnum; i++) {
+ if (strcmp(secnames+sechdrs[i].sh_name, secname) == 0) {
+ if (size)
+ *size = sechdrs[i].sh_size;
+ return (void *)ehdr + sechdrs[i].sh_offset;
+ }
+ }
+ *size = 0;
+ return NULL;
+}
+
+static void * __init find_section64(Elf64_Ehdr *ehdr, const char *secname,
+ unsigned long *size)
+{
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ Elf64_Shdr *sechdrs;
+ unsigned int i;
+ char *secnames;
+
+ /* Grab section headers and strings so we can tell who is who */
+ sechdrs = (void *)ehdr + ehdr->e_shoff;
+ secnames = (void *)ehdr + sechdrs[ehdr->e_shstrndx].sh_offset;
+
+ /* Find the section they want */
+ for (i = 1; i < ehdr->e_shnum; i++) {
+ if (strcmp(secnames+sechdrs[i].sh_name, secname) == 0) {
+ if (size)
+ *size = sechdrs[i].sh_size;
+ return (void *)ehdr + sechdrs[i].sh_offset;
+ }
+ }
+ if (size)
+ *size = 0;
+ return NULL;
+ }
+
+static Elf32_Sym * __init find_symbol32(struct lib32_elfinfo *lib, const char *symname)
+{
+ unsigned int i;
+ char name[32], *c;
+
+ for (i = 0; i < (lib->dynsymsize / sizeof(Elf32_Sym)); i++) {
+ if (lib->dynsym[i].st_name == 0)
+ continue;
+ strncpy(name, lib->dynstr + lib->dynsym[i].st_name, 32);
+ c = strchr(name, '@');
+ if (c)
+ *c = 0;
+ if (strcmp(symname, name) == 0)
+ return &lib->dynsym[i];
+ }
+ return NULL;
+ }
+
+static Elf64_Sym * __init find_symbol64(struct lib64_elfinfo *lib, const char *symname)
+{
+ unsigned int i;
+ char name[32], *c;
+
+ for (i = 0; i < (lib->dynsymsize / sizeof(Elf64_Sym)); i++) {
+ if (lib->dynsym[i].st_name == 0)
+ continue;
+ strncpy(name, lib->dynstr + lib->dynsym[i].st_name, 32);
+ c = strchr(name, '@');
+ if (c)
+ *c = 0;
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ if (strcmp(symname, name) == 0)
+ return &lib->dynsym[i];
+ }
+ return NULL;
+}
+
+/* Note that we assume the section is .text and the symbol is relative to
+ * the library base
+ */
+static unsigned long __init find_function32(struct lib32_elfinfo *lib, const char *symname)
+{
+ Elf32_Sym *sym = find_symbol32(lib, symname);
+
+ if (sym == NULL) {
+ printk(KERN_WARNING "vDSO32: function %s not found !\n", symname);
+ return 0;
+ }
+ return sym->st_value - VDSO32_LBASE;
+}
+
+/* Note that we assume the section is .text and the symbol is relative to
+ * the library base
+ */
+static unsigned long __init find_function64(struct lib64_elfinfo *lib, const char *symname)
+{
+ Elf64_Sym *sym = find_symbol64(lib, symname);
+
+ if (sym == NULL) {
+ printk(KERN_WARNING "vDSO64: function %s not found !\n", symname);
+ return 0;
+ }
+ #ifdef VDS64_HAS_DESCRIPTOR
+ return *((u64 *) (vds64_kbase + sym->st_value - VDSO64_LBASE)) - VDSO64_LBASE;
+ #else
+ return sym->st_value - VDSO64_LBASE;
+ #endif
+}
+
+static __init int vdso_do_find_sections(struct lib32_elfinfo *v32,
+ struct lib64_elfinfo *v64)
+{
+ void *sect;
+
+ /*
+ * Locate symbol tables & text section
+ */
+
+ v32->dynsym = find_section32(v32->hdr, ".dynsym", &v32->dynsymsize);
+ v32->dynstr = find_section32(v32->hdr, ".dynstr", NULL);
+ if (v32->dynsym == NULL || v32->dynstr == NULL) {
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ printk(KERN_ERR "vDSO32: a required symbol section was not found\n");
+ return -1;
+ }
+ sect = find_section32(v32->hdr, ".text", NULL);
+ if (sect == NULL) {
+ printk(KERN_ERR "vDSO32: the .text section was not found\n");
+ return -1;
+ }
+ v32->text = sect - vdso32_kbase;
+
+ v64->dynsym = find_section64(v64->hdr, ".dynsym", &v64->dynsymsize);
+ v64->dynstr = find_section64(v64->hdr, ".dynstr", NULL);
+ if (v64->dynsym == NULL || v64->dynstr == NULL) {
+ printk(KERN_ERR "vDSO64: a required symbol section was not found\n");
+ return -1;
+ }
+ sect = find_section64(v64->hdr, ".text", NULL);
+ if (sect == NULL) {
+ printk(KERN_ERR "vDSO64: the .text section was not found\n");
+ return -1;
+ }
+ v64->text = sect - vdso64_kbase;
+
+ return 0;
+}
+
+static __init void vdso_setup_trampoline(struct lib32_elfinfo *v32,
+ struct lib64_elfinfo *v64)
+{
+ /*
+ * Find signal trampolines
+ */
+
+ vdso64_rt_sigtramp = find_function64(v64, "__kernel_sigtramp_rt64");
+ vdso32_sigtramp = find_function32(v32, "__kernel_sigtramp32");
+ vdso32_rt_sigtramp = find_function32(v32, "__kernel_sigtramp_rt32");
+}
+
+static __init int vdso_fixup_datapage(struct lib32_elfinfo *v32,
+ struct lib64_elfinfo *v64)
+{
+ Elf32_Sym *sym32;
+ Elf64_Sym *sym64;
+
+ sym32 = find_symbol32(v32, "__kernel_datapage_offset");
+ if (sym32 == NULL) {
+ printk(KERN_ERR "vDSO32: Can't find symbol __kernel_datapage_offset !\n");
+ return -1;
+ }
+ *((int *) (vdso32_kbase + (sym32->st_value - VDSO32_LBASE))) =
+ (vdso32_pages << PAGE_SHIFT) - (sym32->st_value - VDSO32_LBASE);
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+
+ sym64 = find_symbol64(v64, "__kernel_datapage_offset");
+ if (sym64 == NULL) {
+ printk(KERN_ERR "vDSO64: Can't find symbol __kernel_datapage_offset !\n");
+ return -1;
+ }
+ *((int*)(vdso64_kbase + sym64->st_value - VDSO64_LBASE)) =
+ (vdso64_pages << PAGE_SHIFT) - (sym64->st_value - VDSO64_LBASE);
+
+ return 0;
+}
+
+static int vdso_do_func_patch32(struct lib32_elfinfo *v32,
+ struct lib64_elfinfo *v64,
+ const char *orig, const char *fix)
+{
+ Elf32_Sym *sym32_gen, *sym32_fix;
+
+ sym32_gen = find_symbol32(v32, orig);
+ if (sym32_gen == NULL) {
+ printk(KERN_ERR "vDSO32: Can't find symbol %s !\n", orig);
+ return -1;
+ }
+ sym32_fix = find_symbol32(v32, fix);
+ if (sym32_fix == NULL) {
+ printk(KERN_ERR "vDSO32: Can't find symbol %s !\n", fix);
+ return -1;
+ }
+ sym32_gen->st_value = sym32_fix->st_value;
+ sym32_gen->st_size = sym32_fix->st_size;
+ sym32_gen->st_info = sym32_fix->st_info;
+ sym32_gen->st_other = sym32_fix->st_other;
+ sym32_gen->st_shndx = sym32_fix->st_shndx;
+
+ return 0;
+}
+
+static int vdso_do_func_patch64(struct lib32_elfinfo *v32,
+ struct lib64_elfinfo *v64,
+ const char *orig, const char *fix)
+{
+ Elf64_Sym *sym64_gen, *sym64_fix;
+
+ sym64_gen = find_symbol64(v64, orig);
+ if (sym64_gen == NULL) {
+ printk(KERN_ERR "vDSO64: Can't find symbol %s !\n", orig);
+ return -1;
+ }
+ sym64_fix = find_symbol64(v64, fix);
+ if (sym64_fix == NULL) {
+ printk(KERN_ERR "vDSO64: Can't find symbol %s !\n", fix);
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```

+ return -1;
+ }
+ sym64_gen->st_value = sym64_fix->st_value;
+ sym64_gen->st_size = sym64_fix->st_size;
+ sym64_gen->st_info = sym64_fix->st_info;
+ sym64_gen->st_other = sym64_fix->st_other;
+ sym64_gen->st_shndx = sym64_fix->st_shndx;
+
+ return 0;
+}
+
+static __init int vdso_fixup_alt_funcs(struct lib32_elfinfo *v32,
+ struct lib64_elfinfo *v64)
+{
+ u32 pvr;
+ int i;
+
+ pvr = mfspr(SPRN_PVR);
+ for (i = 0; i < ARRAY_SIZE(vdso_patches); i++) {
+ struct vdso_patch_def *patch = &vdso_patches[i];
+ int match = (pvr & patch->pvr_mask) == patch->pvr_value;
+
+ DBG("patch %d (mask: %x, pvr: %x) : %s\n",
+ i, patch->pvr_mask, patch->pvr_value, match ? "match" : "skip");
+
+ if (!match)
+ continue;
+
+ DBG("replacing %s with %s...\n", patch->gen_name, patch->fix_name);
+
+ /*
+ * Patch the 32 bits and 64 bits symbols. Note that we do not patch
+ * the "." symbol on 64 bits. It would be easy to do, but doesn't
+ * seem to be necessary, patching the OPD symbol is enough.
+ */
+ vdso_do_func_patch32(v32, v64, patch->gen_name, patch->fix_name);
+ vdso_do_func_patch64(v32, v64, patch->gen_name, patch->fix_name);
+ }
+
+ return 0;
+}
+
+static __init int vdso_setup(void)
+{
+ struct lib32_elfinfo v32;
+ struct lib64_elfinfo v64;
+
+ v32.hdr = vdso32_kbase;
+ v64.hdr = vdso64_kbase;
+

```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ if (vdso_do_find_sections(&v32, &v64))
+ return -1;
+
+ if (vdso_fixup_datapage(&v32, &v64))
+ return -1;
+
+ if (vdso_fixup_alt_funcs(&v32, &v64))
+ return -1;
+
+ vdso_setup_trampoline(&v32, &v64);
+
+ return 0;
+}
+
+void __init vdso_init(void)
+{
+ int i;
+
+ vdso64_pages = (&vdso64_end - &vdso64_start) >> PAGE_SHIFT;
+ vdso32_pages = (&vdso32_end - &vdso32_start) >> PAGE_SHIFT;
+
+ DBG("vdso64_kbase: %p, 0x%x pages, vdso32_kbase: %p, 0x%x pages\n",
+ vdso64_kbase, vdso64_pages, vdso32_kbase, vdso32_pages);
+
+ /*
+ * Initialize the vDSO images in memory, that is do necessary
+ * fixups of vDSO symbols, locate trampolines, etc...
+ */
+ if (vdso_setup()) {
+ printk(KERN_ERR "vDSO setup failure, not enabled !\n");
+ /* XXX should free pages here ? */
+ vdso64_pages = vdso32_pages = 0;
+ return;
+ }
+
+ /* Make sure pages are in the correct state */
+ for (i = 0; i < vdso64_pages; i++) {
+ struct page *pg = virt_to_page(vdso64_kbase + i*PAGE_SIZE);
+ ClearPageReserved(pg);
+ get_page(pg);
+ }
+ for (i = 0; i < vdso32_pages; i++) {
+ struct page *pg = virt_to_page(vdso32_kbase + i*PAGE_SIZE);
+ ClearPageReserved(pg);
+ get_page(pg);
+ }
+ }
+
+int in_gate_area_no_task(unsigned long addr)
+{
+ return 0;
+}
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+}
+
+int in_gate_area(struct task_struct *task, unsigned long addr)
+{
+ return 0;
+}
+
+struct vm_area_struct *get_gate_vma(struct task_struct *tsk)
+{
+ return NULL;
+}
+
```

Index: linux–work/include/asm–ppc64/processor.h

```
=====
--- linux–work.orig/include/asm–ppc64/processor.h 2005–01–31 14:18:44.000000000 +1100
+++ linux–work/include/asm–ppc64/processor.h 2005–01–31 16:25:56.000000000 +1100
@@ –544,8 +544,8 @@
/* This decides where the kernel will search for a free chunk of vm
 * space during mmap's.
 */
–#define TASK_UNMAPPED_BASE_USER32 (PAGE_ALIGN(STACK_TOP_USER32 / 4))
–#define TASK_UNMAPPED_BASE_USER64 (PAGE_ALIGN(STACK_TOP_USER64 / 4))
+#define TASK_UNMAPPED_BASE_USER32 (PAGE_ALIGN(TASK_SIZE_USER32 / 4))
+#define TASK_UNMAPPED_BASE_USER64 (PAGE_ALIGN(TASK_SIZE_USER64 / 4))

#define TASK_UNMAPPED_BASE
((test_thread_flag(TIF_32BIT)||((ppcdebugset(PPCDBG_BINFMT_32ADDR))) ? \
    TASK_UNMAPPED_BASE_USER32 : TASK_UNMAPPED_BASE_USER64 )
@@ –562,7 +562,8 @@
    double fpr[32]; /* Complete floating point set */
    unsigned long fpscr; /* Floating point status (plus pad) */
    unsigned long fpexc_mode; /* Floating–point exception mode */
– unsigned long pad[3]; /* was saved_msr, saved_softe */
+ unsigned long pad[2]; /* was saved_msr, saved_softe */
+ unsigned long vdso_base; /* base of the vDSO library */
#ifdef CONFIG_ALTIVEC
    /* Complete AltiVec register set */
    vector128 vr[32] __attribute__((aligned(16)));
```

Index: linux–work/include/asm–ppc64/systemcfg.h

```
=====
--- linux–work.orig/include/asm–ppc64/systemcfg.h 2005–01–31 15:56:55.000000000 +1100
+++ linux–work/include/asm–ppc64/systemcfg.h 2005–01–31 16:25:56.000000000 +1100
@@ –20,10 +20,14 @@
 * Minor version changes are a hint.
 */
#define SYSTEMCFG_MAJOR 1
–#define SYSTEMCFG_MINOR 0
+#define SYSTEMCFG_MINOR 1

#ifdef __ASSEMBLY__
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```

+#include <linux/unistd.h>
+
+#define SYSCALL_MAP_SIZE ((__NR_syscalls + 31) / 32)
+
struct systemcfg {
    __u8 eye_catcher[16]; /* Eyecatcher: SYSTEMCFG:PPC64 0x00 */
    struct { /* Systemcfg version numbers */
@@ -47,6 +51,8 @@
        __u32 dcache_line_size; /* L1 d-cache line size 0x64 */
        __u32 icache_size; /* L1 i-cache size 0x68 */
        __u32 icache_line_size; /* L1 i-cache line size 0x6C */
+ __u32 syscall_map_64[SYSCALL_MAP_SIZE]; /* map of available syscalls 0x70 */
+ __u32 syscall_map_32[SYSCALL_MAP_SIZE]; /* map of available syscalls */
    };

#ifdef __KERNEL__
Index: linux-work/include/asm-ppc64/a.out.h
=====
--- linux-work.orig/include/asm-ppc64/a.out.h 2005-01-31 14:18:44.000000000 +1100
+++ linux-work/include/asm-ppc64/a.out.h 2005-01-31 16:25:56.000000000 +1100
@@ -30,14 +30,11 @@

#ifdef __KERNEL__

-#define STACK_TOP_USER64 (TASK_SIZE_USER64)
+#define STACK_TOP_USER64 TASK_SIZE_USER64
+#define STACK_TOP_USER32 TASK_SIZE_USER32

-/* Give 32-bit user space a full 4G address space to live in. */
-#define STACK_TOP_USER32 (TASK_SIZE_USER32)
-
-#define STACK_TOP ((test_thread_flag(TIF_32BIT) || \
- (ppcdebugset(PPCDBG_BINFMT_32ADDR))) ? \
- STACK_TOP_USER32 : STACK_TOP_USER64)
+#define STACK_TOP (test_thread_flag(TIF_32BIT) ? \
+ STACK_TOP_USER32 : STACK_TOP_USER64)

#endif /* __KERNEL__ */

Index: linux-work/include/asm-ppc64/elf.h
=====
--- linux-work.orig/include/asm-ppc64/elf.h 2005-01-31 14:18:44.000000000 +1100
+++ linux-work/include/asm-ppc64/elf.h 2005-01-31 16:25:56.000000000 +1100
@@ -238,10 +238,20 @@
/* A special ignored type value for PPC, for glibc compatibility. */
#define AT_IGNOREPPC 22

+/* The vDSO location. We have to use the same value as x86 for glibc's
+ * sake :-))
+ */
+#define AT_SYSINFO_EHDR 33

```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+
extern int dcache_bsize;
extern int icache_bsize;
extern int ucache_bsize;

+/* We do have an arch_setup_additional_pages for vDSO matters */
+#define ARCH_HAS_SETUP_ADDITIONAL_PAGES
+struct linux_binprm;
+extern int arch_setup_additional_pages(struct linux_binprm *bprm, int executable_stack);
+
+/*
+ * The requirements here are:
+ * - keep the final alignment of sp (sp & 0xf)
+@@ -260,6 +270,8 @@
+     NEW_AUX_ENT(AT_DCACHEBSIZE, dcache_bsize); \
+     NEW_AUX_ENT(AT_ICACHEBSIZE, icache_bsize); \
+     NEW_AUX_ENT(AT_UCACHEBSIZE, ucache_bsize); \
+/* vDSO base */ \
+NEW_AUX_ENT(AT_SYSINFO_EHDR, current->thread.vdso_base); \
+} while (0)

+/* PowerPC64 relocations defined by the ABIs */
Index: linux-work/include/asm-ppc64/time.h
=====
--- linux-work.orig/include/asm-ppc64/time.h 2005-01-31 14:18:44.000000000 +1100
+++ linux-work/include/asm-ppc64/time.h 2005-01-31 16:25:56.000000000 +1100
+@@ -43,10 +43,10 @@
+ struct gettimeofday_vars {
+     unsigned long tb_to_xs;
+     unsigned long stamp_xsec;
+ + unsigned long tb_orig_stamp;
+ };

+ struct gettimeofday_struct {
+ - unsigned long tb_orig_stamp;
+     unsigned long tb_ticks_per_sec;
+     struct gettimeofday_vars vars[2];
+     struct gettimeofday_vars * volatile varp;
Index: linux-work/fs/binfmt_elf.c
=====
--- linux-work.orig/fs/binfmt_elf.c 2005-01-31 14:18:24.000000000 +1100
+++ linux-work/fs/binfmt_elf.c 2005-01-31 16:25:56.000000000 +1100
+@@ -772,6 +772,14 @@
+     goto out_free_dentry;
+ }

+#ifndef ARCH_HAS_SETUP_ADDITIONAL_PAGES
+ + retval = arch_setup_additional_pages(bprm, executable_stack);
+ + if (retval < 0) {
+ + send_sig(SIGKILL, current, 0);
+ + goto out_free_dentry;
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ }
+ #endif /* ARCH_HAS_SETUP_ADDITIONAL_PAGES */
+
+     current->mm->start_stack = bprm->p;

+ /* Now we do a little grungy work by mmaping the ELF image into
Index: linux-work/include/asm-ppc64/page.h
=====
--- linux-work.orig/include/asm-ppc64/page.h 2005-01-31 14:18:44.000000000 +1100
+++ linux-work/include/asm-ppc64/page.h 2005-01-31 16:25:56.000000000 +1100
@@ -185,6 +185,9 @@

extern u64 ppc64_pft_size; /* Log 2 of page table size */

+/* We do define AT_SYSINFO_EHDR but don't use the gate mecanism */
+ #define __HAVE_ARCH_GATE_AREA 1
+
+ #endif /* __ASSEMBLY__ */

# ifdef MODULE
Index: linux-work/include/asm-ppc64/vdso.h
=====
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-work/include/asm-ppc64/vdso.h 2005-01-31 16:25:56.000000000 +1100
@@ -0,0 +1,83 @@
+ #ifndef __PPC64_VDSO_H__
+ #define __PPC64_VDSO_H__
+
+ #ifdef __KERNEL__
+
+ /* Default link addresses for the vDSOs */
+ #define VDSO32_LBASE 0
+ #define VDSO64_LBASE 0
+
+ /* Default map addresses */
+ #define VDSO32_MBASE 0x100000
+ #define VDSO64_MBASE 0x100000
+
+ #define VDSO_VERSION_STRING LINUX_2.6.11
+
+ /* Define if 64 bits VDSO has procedure descriptors */
+ #undef VDS64_HAS_DESCRIPTOR
+
+ #ifndef __ASSEMBLY__
+
+ extern unsigned int vdso64_pages;
+ extern unsigned int vdso32_pages;
+
+ /* Offsets relative to thread->vdso_base */
+ extern unsigned long vdso64_rt_sigtramp;
+ extern unsigned long vdso32_sigtramp;
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+extern unsigned long vdso32_rt_sigtramp;
+
+extern void vdso_init(void);
+
+##else /* __ASSEMBLY__ */
+
+##ifdef __VDSO64__
+##ifdef VDS64_HAS_DESCRIPTOR
+##define V_FUNCTION_BEGIN(name) \
+.globl name; \
+.section ".opd","a"; \
+.align 3; \
+name: \
+.quad .name,.TOC.@tocbase,0; \
+.previous; \
+.globl .name; \
+.type .name,@function; \
+.name: \
+
+##define V_FUNCTION_END(name) \
+.size .name,.-.name;
+
+##define V_LOCAL_FUNC(name) (.name)
+
+##else /* VDS64_HAS_DESCRIPTOR */
+##define V_FUNCTION_BEGIN(name) \
+.globl name; \
+name: \
+
+##define V_FUNCTION_END(name) \
+.size name,.-name;
+
+##define V_LOCAL_FUNC(name) (name)
+
+##endif /* VDS64_HAS_DESCRIPTOR */
+##endif /* __VDSO64__ */
+
+##ifdef __VDSO32__
+
+##define V_FUNCTION_BEGIN(name) \
+.globl name; \
+.type name,@function; \
+name: \
+
+##define V_FUNCTION_END(name) \
+.size name,.-name;
+
+##define V_LOCAL_FUNC(name) (name)
+
+##endif /* __VDSO32__ */
```

Linux-Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+
+#endif /* __ASSEMBLY__ */
+
+#endif /* __KERNEL__ */
+
+#endif /* __PPC64_VDSO_H__ */
Index: linux-work/arch/ppc64/mm/init.c
=====
--- linux-work.orig/arch/ppc64/mm/init.c 2005-01-31 14:18:14.000000000 +1100
+++ linux-work/arch/ppc64/mm/init.c 2005-01-31 16:25:56.000000000 +1100
@@ -62,6 +62,7 @@
#include <asm/system.h>
#include <asm/iommu.h>
#include <asm/abs_addr.h>
+#include <asm/vdso.h>

int mem_init_done;
unsigned long ioremap_bot = IMALLOC_BASE;
@@ -743,6 +744,8 @@
#ifdef CONFIG_PPC_ISERIES
    iommu_vio_init();
#endif
+ /* Initialize the vDSO */
+ vdso_init();
}

/*
Index: linux-work/arch/ppc64/kernel/vdso32/gettimeofday.S
=====
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-work/arch/ppc64/kernel/vdso32/gettimeofday.S 2005-01-31 16:25:56.000000000 +1100
@@ -0,0 +1,139 @@
+/*
+ * Userland implementation of gettimeofday() for 32 bits processes in a
+ * ppc64 kernel for use in the vDSO
+ *
+ * Copyright (C) 2004 Benjamin Herrenschmidt (benh@kernel.crashing.org), IBM Corp.
+ *
+ * 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; either version
+ * 2 of the License, or (at your option) any later version.
+ */
+#include <linux/config.h>
+#include <asm/processor.h>
+#include <asm/ppc_asm.h>
+#include <asm/vdso.h>
+#include <asm/offsets.h>
+#include <asm/unistd.h>
+
+.text
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+/*
+ * Exact prototype of gettimeofday
+ *
+ * int __kernel_gettimeofday(struct timeval *tv, struct timezone *tz);
+ *
+ */
+V_FUNCTION_BEGIN(__kernel_gettimeofday)
+.cfi_startproc
+.mflr r12
+.cfi_register lr,r12
+
+.mr r10,r3 /* r10 saves tv */
+.mr r11,r4 /* r11 saves tz */
+.bl __get_datapage@local /* get data page */
+.mr r9, r3 /* datapage ptr in r9 */
+.bl __do_get_xsec@local /* get xsec from tb & kernel */
+.bne- 2f /* out of line -> do syscall */
+
+/* seconds are xsec >> 20 */
+.rlwinm r5,r4,12,20,31
+.rlwimi r5,r3,12,0,19
+.stw r5,TVAL32_TV_SEC(r10)
+
+/* get remaining xsec and convert to usec. we scale
+ * up remaining xsec by 12 bits and get the top 32 bits
+ * of the multiplication
+ */
+.rlwinm r5,r4,12,0,19
+.lis r6,1000000@h
+.ori r6,r6,1000000@l
+.mulhwu r5,r5,r6
+.stw r5,TVAL32_TV_USEC(r10)
+
+.cmpli cr0,r11,0 /* check if tz is NULL */
+.beq 1f
+.lwz r4,CFG_TZ_MINUTEWEST(r9)/* fill tz */
+.lwz r5,CFG_TZ_DSTTIME(r9)
+.stw r4,TZONE_TZ_MINWEST(r11)
+.stw r5,TZONE_TZ_DSTTIME(r11)
+
+.l: mtlr r12
+.blr
+
+.l2: mr r3,r10
+.mr r4,r11
+.li r0,__NR_gettimeofday
+.sc
+.b 1b
+.cfi_endproc
+V_FUNCTION_END(__kernel_gettimeofday)
+
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+/*
+ * This is the core of gettimeofday(), it returns the xsec
+ * value in r3 & r4 and expects the datapage ptr (non clobbered)
+ * in r9. clobbers r0,r4,r5,r6,r7,r8
+*/
+__do_get_xsec:
+.cfi_startproc
+/* Check for update count & load values. We use the low
+ * order 32 bits of the update count
+ */
+1: lwz r8,(CFG_TB_UPDATE_COUNT+4)(r9)
+ andi. r0,r8,1 /* pending update ? loop */
+ bne- 1b
+ xor r0,r8,r8 /* create dependency */
+ add r9,r9,r0
+
+ /* Load orig stamp (offset to TB) */
+ lwz r5,CFG_TB_ORIG_STAMP(r9)
+ lwz r6,(CFG_TB_ORIG_STAMP+4)(r9)
+
+ /* Get a stable TB value */
+2: mftbu r3
+ mftbl r4
+ mftbu r0
+ cmpl cr0,r3,r0
+ bne- 2b
+
+ /* Subtract tb orig stamp. If the high part is non-zero, we jump to the
+ * slow path which call the syscall. If it's ok, then we have our 32 bits
+ * tb_ticks value in r7
+ */
+ subfc r7,r6,r4
+ subfe. r0,r5,r3
+ bne- 3f
+
+ /* Load scale factor & do multiplication */
+ lwz r5,CFG_TB_TO_XS(r9) /* load values */
+ lwz r6,(CFG_TB_TO_XS+4)(r9)
+ mulhwu r4,r7,r5
+ mulhwu r6,r7,r6
+ mullw r6,r7,r5
+ addc r6,r6,r0
+
+ /* At this point, we have the scaled xsec value in r4 + XER:CA
+ * we load & add the stamp since epoch
+ */
+ lwz r5,CFG_STAMP_XSEC(r9)
+ lwz r6,(CFG_STAMP_XSEC+4)(r9)
+ adde r4,r4,r6
+ addze r3,r5
+
```

Linux–Kernel: [PATCH] ppc64: Implement a vDSO and use it for signal trampoline

```
+ /* We now have our result in r3,r4. We create a fake dependency
+ * on that result and re-check the counter
+ */
+ xor r0,r4,r4
+ add r9,r9,r0
+ lwz r0,(CFG_TB_UPDATE_COUNT+4)(r9)
+ cmpl cr0,r8,r0 /* check if updated */
+ bne- 1b
+
+ /* Warning ! The caller expects CR:EQ to be set to indicate a
+ * successful calculation (so it won't fallback to the syscall
+ * method). We have overridden that CR bit in the counter check,
+ * but fortunately, the loop exit condition _is_ CR:EQ set, so
+ * we can exit safely here. If you change this code, be careful
+ * of that side effect.
+ */
+3: blr
+ .cfi_endproc
Index: linux-work/arch/ppc64/kernel/vdso32/sigtramp.S
=====
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-work/arch/ppc64/kernel/vdso32/sigtramp.S 2005-01-31 16:25:56.000000000 +1100
@@ -0,0 +1,300 @@
+/*
+ * Signal trampolines for 32 bits processes in a ppc64 kernel for
+ * use in the vDSO
+ *
+ * Copyright (C) 2004 Benjamin Herrenschmidt (benh@kernel.crashing.org), IBM Corp.
+ * Copyright (C) 2004 Alan Modra (amodra@au.ibm.com)), IBM Corp.
+ *
+ * This program is free software; you can redistribute it and/or
+ * modify it under the terms of the GNU General Public Li
```