

[PATCH 3/11] perfmon2 patch for review: new generic header files

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-05/msg03098.html>

- *From:* Stephane Eranian <eranian@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 12 May 2006 09:33:44 -0700
-

This patch contains the new generic header files.

```
--- linux-2.6.17-rc4.orig/include/linux/perfmon.h 1969-12-31 16:00:00.000000000 -0800
+++ linux-2.6.17-rc4/include/linux/perfmon.h 2006-05-12 03:18:52.000000000 -0700
@@ -0,0 +1,704 @@
+/*
+ * Copyright (c) 2001-2006 Hewlett-Packard Development Company, L.P.
+ * Contributed by Stephane Eranian <eranian@xxxxxxxxxx>
+ */
+
+#ifndef __LINUX_PERFMON_H__
+#define __LINUX_PERFMON_H__
+
+#ifdef CONFIG_PERFMON
+
+/*
+ * user interface definitions
+ *
+ * Do not use directly for applications, use libpfm/libc provided
+ * header file instead.
+ */
+
+#define PFM_MAX_HW_PMCS 256
+#define PFM_MAX_HW_PMDS 256
+#define PFM_MAX_XTRA_PMCS 64
+#define PFM_MAX_XTRA_PMDS 64
+
+#define PFM_MAX_PMCS (PFM_MAX_HW_PMCS+PFM_MAX_XTRA_PMCS)
+#define PFM_MAX_PMDS (PFM_MAX_HW_PMDS+PFM_MAX_XTRA_PMDS)
+
+/*
+ * number of elements for each type of bitvector
+ * all bitvectors use u64 fixed size type on all architectures.
+ */
+#define PFM_BVSIZE(x) (((x)+(sizeof(u64)<<3)-1) / (sizeof(u64)<<3))
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+ #define PFM_HW_PMD_BV PFM_BVSIZE(PFM_MAX_HW_PMDS)
+ #define PFM_PMD_BV PFM_BVSIZE(PFM_MAX_PMDS)
+ #define PFM_PMC_BV PFM_BVSIZE(PFM_MAX_PMCS)
+
+ /*
+  * custom sampling buffer identifier type
+  */
+ typedef __u8 pfm_uuid_t[16];
+
+ /*
+  * PMC/PMD flags to use with pfm_write_pmds() or pfm_write_pmcs()
+  *
+  * reg_flags layout:
+  * bit 00–15 : generic flags
+  * bit 16–23 : arch-specific flags
+  * bit 24–31 : error codes
+  */
+ #define PFM_REGFL_OVFL_NOTIFY 0x1 /* PMD: send notification on overflow */
+ #define PFM_REGFL_RANDOM 0x2 /* PMD: randomize sampling interval */
+ #define PFM_REGFL_NO_EMUL64 0x4 /* PMC: no 64-bit emulation for counter */
+
+ /*
+  * event set flags layout:
+  * bit 00–15 : generic flags
+  * bit 16–31 : arch-specific flags
+  */
+ #define PFM_SETFL_OVFL_SWITCH 0x01 /* enable switch on overflow */
+ #define PFM_SETFL_TIME_SWITCH 0x02 /* switch set on timeout */
+ #define PFM_SETFL_EXPL_NEXT 0x04 /* use set_id_next as the next set */
+ #define PFM_SETFL_EXCL_IDLE 0x08 /* exclude idle task (syswide only) */
+
+ /*
+  * PMD/PMC return flags in case of error (ignored on input)
+  *
+  * reg_flags layout:
+  * bit 00–15 : generic flags
+  * bit 16–23 : arch-specific flags
+  * bit 24–31 : error codes
+  *
+  * Those flags are used on output and must be checked in case EINVAL is
+  * returned by a command accepting a vector of values and each has a flag
+  * field, such as pfarg_pmc or pfarg_pmd.
+  */
+ #define PFM_REG_RETFL_NOTAVAIL (1<<31) /* implemented but not available */
+ #define PFM_REG_RETFL_EINVAL (1<<30) /* entry is invalid */
+ #define PFM_REG_RETFL_NOSET (1<<29) /* event set does not exist */
+ #define PFM_REG_RETFL_MASK (PFM_REG_RETFL_NOTAVAIL|
+ PFM_REG_RETFL_EINVAL|
+ PFM_REG_RETFL_NOSET)
+
+ #define PFM_REG_HAS_ERROR(flag) (((flag) & PFM_REG_RETFL_MASK) != 0)
```

```

+
+typedef __u32 __bitwise pfm_flags_t;
+/*
+ * Request structure used to define a context
+ */
+struct pfarg_ctx {
+ pfm_uuid_t ctx_smpl_buf_id; /* which buffer format to use */
+ pfm_flags_t ctx_flags; /* noblock/block/syswide */
+ __s32 ctx_fd; /* ret arg: fd for context */
+ __u64 ctx_smpl_buf_size; /* ret arg: actual buffer size */
+ __u64 ctx_reserved3[12]; /* for future use */
+};
+/*
+ * context flags (ctx_flags)
+ *
+ */
+#define PFM_FL_NOTIFY_BLOCK 0x01 /* block task on user notifications */
+#define PFM_FL_SYSTEM_WIDE 0x02 /* create a system wide context */
+#define PFM_FL_OVFL_NO_MSG 0x80 /* no overflow msgs */
+#define PFM_FL_MAP_SETS 0x10 /* event sets are remapped */
+
+
+/*
+ * argument structure for pfm_write_pmcs()
+ */
+struct pfarg_pmc {
+ __u16 reg_num; /* which register */
+ __u16 reg_set; /* event set for this register */
+ pfm_flags_t reg_flags; /* input: flags, return: reg error */
+ __u64 reg_value; /* pmc value */
+ __u64 reg_reserved2[4]; /* for future use */
+};
+
+/*
+ * argument structure for pfm_write_pmds() and pfm_read_pmds()
+ */
+struct pfarg_pmd {
+ __u16 reg_num; /* which register */
+ __u16 reg_set; /* event set for this register */
+ pfm_flags_t reg_flags; /* input: flags, return: reg error */
+ __u64 reg_value; /* initial pmc/pmd value */
+ __u64 reg_long_reset; /* value to reload after notification */
+ __u64 reg_short_reset; /* reset after counter overflow */
+ __u64 reg_last_reset_val; /* return: PMD last reset value */
+ __u64 reg_ovfl_switch_cnt; /* #overflows before switch */
+ __u64 reg_reset_pmds[PFM_PMD_BV]; /* reset on overflow */
+ __u64 reg_smpl_pmds[PFM_PMD_BV]; /* record in sample */
+ __u64 reg_smpl_eventid; /* opaque event identifier */
+ __u64 reg_random_mask; /* bitmask used to limit random value */
+ __u32 reg_random_seed; /* seed for randomization */
+ __u32 reg_reserved2[7]; /* for future use */

```

```

+};
+
+/*
+ * optional argument to pfm_start(), pass NULL if no arg needed
+ */
+struct pfarg_start {
+ __u16 start_set; /* event set to start with */
+ __u16 start_reserved1; /* for future use */
+ __u32 start_reserved2; /* for future use */
+ __u64 reserved3[3]; /* for future use */
+};
+
+/*
+ * argument to pfm_load_context()
+ */
+struct pfarg_load {
+ __u32 load_pid; /* thread to attach to */
+ __u16 load_set; /* set to load first */
+ __u16 load_reserved1; /* for future use */
+ __u64 load_reserved2[3]; /* for future use */
+};
+
+/*
+ * argument to pfm_create_evtsets()/pfm_delete_evtsets()
+ *
+ * max timeout: 1h11mn33s (2<<32 usecs)
+ */
+struct pfarg_setdesc {
+ __u16 set_id; /* which set */
+ __u16 set_id_next; /* next set to go to */
+ pfm_flags_t set_flags; /* input: flags, return: err flag */
+ __u32 set_timeout; /* req/eff switch timeout in usecs */
+ __u32 set_reserved1; /* for future use */
+ __u64 set_mmap_offset; /* ret arg: cookie for mmap offset */
+ __u64 reserved[5]; /* for future use */
+};
+
+/*
+ * argument to pfm_getinfo_evtsets()
+ */
+struct pfarg_setinfo {
+ __u16 set_id; /* which set */
+ __u16 set_id_next; /* out: next set to go to */
+ pfm_flags_t set_flags; /* out: flags or error */
+ __u64 set_ovfl_pmds[PFM_PMD_BV]; /* out: last ovfl PMDs */
+ __u64 set_runs; /* out: #times the set was active */
+ __u32 set_timeout; /* out: effective switch timeout in usecs */
+ __u32 set_reserved1; /* for future use */
+ __u64 set_act_duration; /* out: time set active (cycles) */
+ __u64 set_mmap_offset; /* cookie to for mmap offset */
+ __u64 reserved[4]; /* for future use */

```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+};
+
+/*
+ * default value for the user and group security parameters in
+ * /proc/sys/kernel/perfmon/sys_group
+ * /proc/sys/kernel/perfmon/task_group
+ */
+#define PFM_GROUP_PERM_ANY -1 /* any user/group */
+
+/*
+ * remapped set view
+ *
+ * IMPORTANT: cannot be bigger than PAGE_SIZE
+ */
+struct pfm_set_view {
+ __u32 set_status; /* set status: active/inact */
+ __u32 set_reserved1; /* for future use */
+ __u64 set_runs; /* number of activations */
+ __u64 set_pmids[PFM_MAX_PMDS]; /* 64-bit value of PMDS */
+ volatile unsigned long set_seq; /* sequence number of updates */
+};
+
+/*
+ * pfm_set_view status flags
+ */
+#define PFM_SETVFL_ACTIVE 0x1 /* set is active */
+
+struct pfm_ovfl_msg {
+ __u32 msg_type; /* generic message header */
+ __u32 msg_ovfl_pid; /* process id */
+ __u64 msg_ovfl_pmids[PFM_HW_PMD_BV]; /* overflowed PMDs */
+ __u16 msg_active_set; /* active set at overflow */
+ __u16 msg_ovfl_cpu; /* cpu of PMU interrupt */
+ __u32 msg_ovfl_tid; /* kernel thread id */
+ __u64 msg_ovfl_ip; /* IP on PMU intr */
+};
+
+#define PFM_MSG_OVFL 1 /* an overflow happened */
+#define PFM_MSG_END 2 /* task to which context was attached ended */
+
+union pfm_msg {
+ __u32 type;
+ struct pfm_ovfl_msg pfm_ovfl_msg;
+};
+
+/*
+ * perfmon version number
+ */
+#define PFM_VERSION_MAJ 2U
+#define PFM_VERSION_MIN 2U
+#define PFM_VERSION (((PFM_VERSION_MAJ&0xffff)<<16)|\
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+ (PFM_VERSION_MIN & 0xffff)
+#define PFM_VERSION_MAJOR(x) (((x)>>16) & 0xffff)
+#define PFM_VERSION_MINOR(x) ((x) & 0xffff)
+
+
+/*
+ * This part of the header file is meant for kernel level code only including
+ * kernel modules
+ */
+#ifdef __KERNEL__
+
+#include <linux/file.h>
+#include <linux/seq_file.h>
+#include <linux/interrupt.h>
+#include <linux/kobject.h>
+
+/*
+ * perfmon context state
+ */
+#define PFM_CTX_UNLOADED 1 /* context is not loaded onto any task */
+#define PFM_CTX_LOADED 2 /* context is loaded onto a task */
+#define PFM_CTX_MASKED 3 /* context is loaded, monitoring is masked */
+#define PFM_CTX_ZOMBIE 4 /* context lost owner but is still attached */
+
+/*
+ * depth of message queue
+ */
+#define PFM_MAX_MSGS 8
+#define PFM_CTXQ_EMPTY(g) ((g)->msgq_head == (g)->msgq_tail)
+
+/*
+ * type of PMD reset for pfm_reset_pmds() or pfm_switch_sets()
+ */
+#define PFM_PMD_RESET_NONE 0 /* do not reset (pfm_switch_set) */
+#define PFM_PMD_RESET_SHORT 1 /* use short reset value */
+#define PFM_PMD_RESET_LONG 2 /* use long reset value */
+
+/*
+ * describe the content of the pfm_syst_info field
+ */
+#define PFM_CPUINFO_TIME_SWITCH 0x20 /* current set is time-switched */
+
+struct pfm_controls {
+ int debug; /* debugging via syslog */
+ int debug_ovfl; /* overflow handling debugging */
+ int expert_mode; /* PMC/PMD value checking */
+ gid_t sys_group; /* gid to create a syswide context */
+ gid_t task_group; /* gid to create a per-task context */
+ size_t arg_size_max; /* maximum vector argument size */
+ size_t simpl_buf_size_max; /* max buf mem, -1 for infinity */
+};
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+
+DECLARE_PER_CPU(struct task_struct *, pmu_owner);
+DECLARE_PER_CPU(struct pfm_context *, pmu_ctx);
+DECLARE_PER_CPU(unsigned long, pfm_syst_info);
+DECLARE_PER_CPU(u64, pmu_activation_number);
+DECLARE_PER_CPU(struct pfm_stats, pfm_stats);
+
+/*
+ * logging
+ */
+#define PFM_ERR(f, x...) printk(KERN_ERR "perfmon: " f "\n", ## x)
+#define PFM_WARN(f, x...) printk(KERN_WARNING "perfmon: " f "\n", ## x)
+#define PFM_LOG(f, x...) printk(KERN_NOTICE "perfmon: " f "\n", ## x)
+#define PFM_INFO(f, x...) printk(KERN_INFO "perfmon: " f "\n", ## x)
+
+/*
+ * debugging
+ */
+#define PFM_DEBUGGING 1
+#ifdef PFM_DEBUGGING
+#define PFM_DBG(f, x...) \
+ do { \
+ if (unlikely(pfm_controls.debug >0)) { \
+ printk("perfmon: %s.%d: CPU%d [%d]: " f "\n", \
+ __FUNCTION__, __LINE__, \
+ smp_processor_id(), current->pid, ## x); \
+ } \
+ } while (0)
+
+#define PFM_DBG_ovfl(f, x...) \
+ do { \
+ if (unlikely(pfm_controls.debug_ovfl >0)) { \
+ printk("perfmon: %s.%d: CPU%d [%d]: " f "\n", \
+ __FUNCTION__, __LINE__, \
+ smp_processor_id(), current->pid, ## x); \
+ } \
+ } while (0)
+#else
+#define PFM_DBG(f, x...) do { } while(0)
+#define PFM_DBG_ovfl(f, x...) do { } while(0)
+#endif
+
+/*
+ * global information about all sessions
+ * mostly used to synchronize between system wide and per-process
+ */
+struct pfm_sessions {
+ u32 pfs_task_sessions; /* #num loaded per-thread sessions */
+ u32 pfs_sys_sessions; /* #num loaded system wide sessions */
+ size_t pfs_cur_smpl_buf_mem; /* current smpl buf mem usage */
+ cpumask_t pfs_sys_cpumask; /* bitmask of used cpus */

```

```

+};
+
+/*
+ * PMD information
+ * software maintained value is in the pfm_set_view structure.
+ */
+struct pfm_pmd {
+ u64 lval; /* last reset value */
+ u64 ovflsw_thres; /* #overflows left before switching */
+ u64 long_reset; /* reset value on sampling overflow */
+ u64 short_reset; /* reset value on overflow */
+ u64 reset_pmds[PFM_PMD_BV]; /* pmcs to reset on overflow */
+ u64 smpl_pmds[PFM_PMD_BV]; /* pmcs to record on overflow */
+ u64 mask; /* mask for generator */
+ u64 seed; /* seed for generator (must be 64 bits here) */
+ u32 flags; /* notify/do not notify */
+ u64 ovflsw_ref_thres; /* #overflows before switching to next set */
+ u64 eventid; /* overflow event identifier */
+};
+
+/*
+ * perfmon context: encapsulates all the state of a monitoring session
+ */
+struct pfm_event_set {
+ u16 id;
+ u16 id_next; /* which set to go to from this one */
+ pfm_flags_t flags; /* public set flags */
+
+
+ struct pfm_event_set *next; /* next in the ordered list */
+ struct pfm_event_set *switch_next; /* address of set to go to */
+ pfm_flags_t priv_flags; /* private flags */
+ u32 npend_ovfls; /* number of pending PMD overflow */
+
+
+ u64 used_pmds[PFM_PMD_BV]; /* used PMDs */
+ u64 povfl_pmds[PFM_PMD_BV]; /* pending overflowed PMDs */
+ u64 ovfl_pmds[PFM_PMD_BV]; /* last overflowed PMDs */
+ u64 reset_pmds[PFM_PMD_BV]; /* PMDs to reset */
+ u64 ovfl_notify[PFM_PMD_BV]; /* notify on overflow */
+ u64 pmcs[PFM_MAX_PMCS]; /* PMC values */
+
+
+ u16 nused_pmds; /* max number of used PMDs */
+ u16 nused_pmcs; /* max number of used PMCs */
+
+
+ struct pfm_pmd pmcs[PFM_MAX_PMDS]; /* 64-bit SW PMDs */
+ struct pfm_set_view *view; /* pointer to view */
+ u64 switch_timeout; /* switch timeout */
+ u64 timeout; /* timeout remaining */
+ u64 duration_start; /* start cycles */
+ u64 duration; /* total active cycles */
+ off_t mmap_offset; /* view mmap offset */
+ u64 used_pmcs[PFM_PMC_BV]; /* used PMCs (keep for arbitration) */

```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+
+ unsigned long last_iip; /* last interrupt instruction pointer */
+ u64 last_ovfl_pmd_reset; /* reset of lowest idx of last overflowed pmcs */
+ unsigned int last_ovfl_pmd; /* lowest idx of last overflowed pmcs */
+};
+
+/*
+ * common private event set flags (priv_flags)
+ *
+ * upper 16 bits: for arch-specific use
+ * lower 16 bits: for common use
+ */
+#define PFM_SETFL_PRIV_MOD_PMDS 0x1 /* PMD register(s) modified */
+#define PFM_SETFL_PRIV_MOD_PMCS 0x2 /* PMC register(s) modified */
+#define PFM_SETFL_PRIV_SWITCH 0x4 /* must switch set on restart */
+#define PFM_SETFL_PRIV_MOD_BOTH (PFM_SETFL_PRIV_MOD_PMDS |
PFM_SETFL_PRIV_MOD_PMCS)
+
+/*
+ * context flags
+ */
+struct pfm_context_flags {
+ unsigned int block:1; /* task blocks on user notifications */
+ unsigned int system:1; /* do system wide monitoring */
+ unsigned int excl_idle:1; /* exclude idle task (syswide) */
+ unsigned int no_msg:1; /* no message sent on overflow */
+ unsigned int can_restart:1; /* allowed to issue a PFM_RESTART */
+ unsigned int switch_ovfl:1; /* switch set on counter ovfl */
+ unsigned int switch_time:1; /* switch set on timeout */
+ unsigned int mapset:1; /* event sets are remapped */
+ unsigned int started:1; /* pfm_start() issued */
+ unsigned int trap_reason:2; /* reason for pfm_handle_work() */
+ unsigned int reserved:21; /* for future use */
+};
+
+/*
+ * values for trap_reason
+ */
+#define PFM_TRAP_REASON_NONE 0x0 /* nothing to do */
+#define PFM_TRAP_REASON_BLOCK 0x1 /* block on overflow */
+#define PFM_TRAP_REASON_RESET 0x2 /* reset PMDs */
+#define PFM_TRAP_REASON_ZOMBIE 0x3 /* cleanup because of ZOMBIE */
+
+/*
+ * check_mask bitmask values for pfm_check_task_state()
+ */
+#define PFM_CMD_STOPPED 0x01 /* command needs thread stopped */
+#define PFM_CMD_UNLOADED 0x02 /* command needs ctx unloaded */
+#define PFM_CMD_UNLOAD 0x04 /* command is unload */
+
+#include <linux/perfmon_pmu.h>
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+#include <linux/perfmon_fmt.h>
+#include <linux/perfmon_kernel.h>
+
+/*
+ * perfmon context: encapsulates all the state of a monitoring session
+ */
+struct pfm_context {
+ spinlock_t lock; /* context protection */
+
+ struct file *filp; /* filp */
+
+ struct pfm_context_flags flags; /* flags */
+ u32 state; /* state */
+ struct task_struct *task; /* attached task */
+
+ struct completion restart_complete; /* block on notification */
+ u64 duration_start; /* last cycles at last activation */
+ u64 duration; /* total cycles context was active */
+ u64 last_act; /* last activation */
+ u32 last_cpu; /* last CPU used (SMP only) */
+ u32 cpu; /* cpu bound to context */
+ struct pfm_smpl_fmt *smpl_fmt; /* buffer format callbacks */
+ void *smpl_addr; /* smpl buffer base */
+ size_t smpl_size;
+ wait_queue_head_t msgq_wait;
+ union pfm_msg msgq[PFM_MAX_MSGS];
+ int msgq_head;
+ int msgq_tail;
+ struct fasync_struct *async_queue;
+
+ struct pfm_event_set *active_set; /* active set */
+ struct pfm_event_set *sets; /* ordered list of sets */
+
+ /*
+ * save stack space by allocating temporary variables for
+ * pfm_overflow_handler() in pfm_context
+ */
+ struct pfm_ovfl_arg ovfl_arg;
+ u64 ovfl_ovfl_notify[PFM_PMD_BV];
+};
+
+#define pfm_ctx_arch(c) ((struct pfm_arch_context *)((c)+1))
+
+static inline void pfm_set_pmu_owner(struct task_struct *task,
+ struct pfm_context *ctx)
+{
+ BUG_ON(task && task->pid == 0);
+ __get_cpu_var(pmu_owner) = task;
+ __get_cpu_var(pmu_ctx) = ctx;
+}
+
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+#ifdef CONFIG_SMP
+static inline void pfm_inc_activation(void)
+{
+ __get_cpu_var(pmu_activation_number)++;
+}
+static inline void pfm_set_activation(struct pfm_context *ctx)
+{
+ ctx->last_act = __get_cpu_var(pmu_activation_number);
+}
+static inline void pfm_set_last_cpu(struct pfm_context *ctx, int cpu)
+{
+ ctx->last_cpu = cpu;
+}
+#else
+#define pfm_inc_activation() do { } while(0)
+#define pfm_set_activation(c) do { } while(0)
+#define pfm_set_last_cpu(c, p) do { } while(0)
+#endif
+
+static inline void pfm_modview_begin(struct pfm_event_set *set)
+{
+ set->view->set_seq++;
+}
+
+static inline void pfm_modview_end(struct pfm_event_set *set)
+{
+ set->view->set_seq++;
+}
+
+static inline void pfm_retflag_set(u32 flags, u32 val)
+{
+ flags &= ~PFM_REG_RETFL_MASK;
+ flags |= (val);
+}
+
+extern struct _pfm_pmu_config *pfm_pmu_conf;
+extern struct pfm_controls pfm_controls;
+
+int pfm_get_args(void __user *arg, size_t sz, void **kargs);
+int pfm_get_smpl_arg(pfm_uid_t uuid, void __user *uaddr, size_t usz,
+ void **arg, struct pfm_smpl_fmt **);
+
+void pfm_undo_create_context(int fd, struct pfm_context *ctx);
+int pfm_alloc_fd(struct file **cfile);
+void pfm_syswide_cleanup_other_cpu(struct pfm_context *ctx);
+
+int __pfm_write_pmcs(struct pfm_context *, struct pfarg_pmc *, int);
+int __pfm_write_pmds(struct pfm_context *, struct pfarg_pmd *, int, int);
+int __pfm_read_pmds(struct pfm_context *, struct pfarg_pmd *, int);
+void __pfm_reset_stats(void);
+int __pfm_load_context(struct pfm_context *ctx, struct pfarg_load *req);
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+int __pfm_unload_context(struct pfm_context *ctx, int defer_release);
+int __pfm_stop(struct pfm_context *ctx);
+int __pfm_restart(struct pfm_context *ctx);
+int __pfm_start(struct pfm_context *ctx, struct pfarg_start *start);
+int __pfm_delete_evtsets(struct pfm_context *ctx, void *arg, int count);
+int __pfm_getinfo_evtsets(struct pfm_context *ctx, struct pfarg_setinfo *req,
+ int count);
+int __pfm_create_evtsets(struct pfm_context *ctx, struct pfarg_setdesc *req,
+ int count);
+int __pfm_create_context(struct pfarg_ctx *, struct pfm_smpl_fmt *, void *, int,
+ struct pfm_context **);
+int pfm_check_task_state(struct pfm_context *ctx, int cmd,
+ unsigned long *flags);
+
+struct pfm_event_set *pfm_find_set(struct pfm_context *ctx, u16 set_id,
+ int alloc);
+
+struct pfm_context * pfm_get_ctx(int fd);
+
+void pfm_context_free(struct pfm_context *ctx);
+struct pfm_context *pfm_context_alloc(void);
+
+
+int pfm_pmu_conf_get(int);
+void pfm_pmu_conf_put(void);
+
+int pfm_reserve_session(struct pfm_context *ctx, u32 cpu);
+int pfm_release_session(struct pfm_context *ctx, u32 cpu);
+
+int pfm_smpl_buffer_alloc(struct pfm_context *ctx, size_t rsize);
+int pfm_reserve_buf_space(size_t size);
+void pfm_release_buf_space(size_t size);
+
+struct pfm_smpl_fmt *pfm_smpl_fmt_get(pfm_uid_t uuid);
+void pfm_smpl_fmt_put(struct pfm_smpl_fmt *fmt);
+int pfm_use_smpl_fmt(pfm_uid_t uuid);
+
+int pfm_sysfs_init(void);
+ssize_t pfm_sysfs_session_show(char *buf, size_t , int);
+int pfm_sysfs_remove_pmu(struct _pfm_pmu_config *pmu);
+int pfm_sysfs_add_pmu(struct _pfm_pmu_config *pmu);
+
+int pfm_sysfs_remove_fmt(struct pfm_smpl_fmt *fmt);
+int pfm_sysfs_add_fmt(struct pfm_smpl_fmt *fmt);
+
+irqreturn_t pfm_interrupt_handler(int, void *, struct pt_regs *);
+void pfm_save_pmds_release(struct pfm_context *ctx);
+
+void pfm_reset_pmds(struct pfm_context *ctx, struct pfm_event_set *set,
+ int reset_mode);
+void pfm_switch_sets(struct pfm_context *ctx,
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+ struct pfm_event_set *new_set,
+ int reset_mode,
+ int no_restart);
+
+void pfm_mask_monitoring(struct pfm_context *ctx);
+int pfm_ovfl_notify_user(struct pfm_context *ctx,
+ struct pfm_event_set *set,
+ unsigned long ip);
+
+int init_pfm_fs(void);
+int pfm_is_fd(struct file *filp);
+
+u64 carta_random32 (u64 seed);
+
+void pfm_resume_after_ovfl(struct pfm_context *ctx);
+
+int __pfm_close(struct pfm_context *ctx, struct file *filp);
+
+static inline void pfm_put_ctx(struct pfm_context *ctx)
+{
+ fput(ctx->filp);
+}
+
+#define PFM_MAX_NUM_SETS 65536
+#define PFM_SET_REMAP_SCALAR PAGE_SIZE
+#define PFM_SET_REMAP_OFFSETS 16384 /* number of pages to offset */
+#define PFM_SET_REMAP_BASE (PFM_SET_REMAP_OFFSETS*PAGE_SIZE)
+#define PFM_SET_REMAP_OFFSETS_MAX (PFM_SET_REMAP_OFFSETS+\
+ PFM_MAX_NUM_SETS*PFM_SET_REMAP_SCALAR)
+
+#define PFM_ONE_64 ((u64)1)
+
+struct pfm_stats {
+ u64 pfm_ovfl_intr_replay_count; /* replayed ovfl interrupts */
+ u64 pfm_ovfl_intr_regular_count; /* processed ovfl interrupts */
+ u64 pfm_ovfl_intr_all_count; /* total ovfl interrupts */
+ u64 pfm_ovfl_intr_cycles; /* cycles in ovfl interrupts */
+ u64 pfm_ovfl_intr_phase1; /* cycles in ovfl interrupts */
+ u64 pfm_ovfl_intr_phase2; /* cycles in ovfl interrupts */
+ u64 pfm_ovfl_intr_phase3; /* cycles in ovfl interrupts */
+ u64 pfm_fmt_handler_calls; /* # calls smpl buffer handler */
+ u64 pfm_fmt_handler_cycles; /* cycle in smpl format handler */
+ u64 pfm_set_switch_count; /* #set_switches on this CPU */
+ u64 pfm_set_switch_cycles; /* cycles for switching sets */
+ u64 pfm_handle_timeout_count; /* #count of set timeouts handled */
+ struct kobject kobj; /* for sysfs internal use only */
+};
+#define to_stats(n) container_of(n, struct pfm_stats, kobj)
+
+
+#include <asm/perfmon.h>
```

```

+
+ #define PFM_BPL 64
+ #define PFM_LBPL 6 /* log2(BPL) */
+
+ /*
+ * those operations are not provided by linux/bitmap.h.
+ * We do not need atomicity nor volatile accesses here.
+ * All bitmaps are 64-bit wide.
+ */
+ static inline void pfm_bv_set(u64 *bv, unsigned int rnum)
+ {
+   bv[rnum >> PFM_LBPL] |= PFM_ONE_64 << (rnum & (PFM_BPL - 1));
+ }
+
+ static inline int pfm_bv_isset(u64 *bv, unsigned int rnum)
+ {
+   return bv[rnum >> PFM_LBPL] & (PFM_ONE_64 << (rnum & (PFM_BPL - 1))) ? 1 : 0;
+ }
+
+ static inline void pfm_bv_clear(u64 *bv, unsigned int rnum)
+ {
+   bv[rnum >> PFM_LBPL] &= ~(PFM_ONE_64 << (rnum & (PFM_BPL - 1)));
+ }
+
+ /*
+ * read a single PMD register. PMD register mapping is provided by PMU
+ * description module. Some PMD registers are require a special read
+ * handler (e.g., virtual PMD mapping to a SW resource).
+ */
+ static inline u64 pfm_read_pmd(struct pfm_context *ctx, unsigned int cnum)
+ {
+   if (pfm_pmu_conf->pmd_desc[cnum].type & PFM_REG_V)
+     return pfm_pmu_conf->pmd_sread(ctx, cnum);
+
+   return pfm_arch_read_pmd(ctx, cnum);
+ }
+
+ static inline void pfm_write_pmd(struct pfm_context *ctx, unsigned int cnum, u64 value)
+ {
+   /*
+    * PMD writes are ignored for read-only registers
+    */
+   if (pfm_pmu_conf->pmd_desc[cnum].type & PFM_REG_RO)
+     return;
+
+   if (pfm_pmu_conf->pmd_desc[cnum].type & PFM_REG_V) {
+     pfm_pmu_conf->pmd_swrite(ctx, cnum, value);
+     return;
+   }
+   pfm_arch_write_pmd(ctx, cnum, value);
+ }

```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+
+#define ulp(_x) ((unsigned long *)_x)
+
+#define PFM_NORMAL 0
+#define PFM_COMPAT 1
+#define PFM_KAPI 2
+
+#endif /* __KERNEL__ */
+
+#endif /* CONFIG_PERFMON */
+
+#endif /* __LINUX_PERFMON_H__ */
--- linux-2.6.17-rc4.orig/include/linux/perfmon_dfl_smpl.h 1969-12-31 16:00:00.000000000 -0800
+++ linux-2.6.17-rc4/include/linux/perfmon_dfl_smpl.h 2006-05-12 03:18:52.000000000 -0700
@@ -0,0 +1,68 @@
+/*
+ * Copyright (c) 2005-2006 Hewlett-Packard Development Company, L.P.
+ * Contributed by Stephane Eranian <eranian@xxxxxxxxxxx>
+ *
+ * This file implements the new dfl sampling buffer format
+ * for perfmon2 subsystem.
+ */
+#ifndef __PERFMON_DFL_SMPL_H__
+#define __PERFMON_DFL_SMPL_H__ 1
+
+#define PFM_DFL_SMPL_UUID { \
+ 0xd1, 0x39, 0xb2, 0x9e, 0x62, 0xe8, 0x40, 0xe4, \
+ 0xb4, 0x02, 0x73, 0x07, 0x87, 0x92, 0xe9, 0x37 }
+
+/*
+ * format specific parameters (passed at context creation)
+ */
+struct pfm_dfl_smpl_arg {
+  __u64 buf_size; /* size of the buffer in bytes */
+  __u32 buf_flags; /* buffer specific flags */
+  __u32 reserved1; /* for future use */
+  __u64 reserved[6]; /* for future use */
+};
+
+/*
+ * This header is at the beginning of the sampling buffer returned to the user.
+ * It is directly followed by the first record.
+ */
+struct pfm_dfl_smpl_hdr {
+  __u64 hdr_count; /* how many valid entries */
+  __u64 hdr_cur_offs; /* current offset from top of buffer */
+  __u64 hdr_overflows; /* #overflows for buffer */
+  __u64 hdr_buf_size; /* bytes in the buffer */
+  __u64 hdr_min_buf_space; /* minimal buffer size (internal use) */
+  __u32 hdr_version; /* smpl format version */
+  __u32 hdr_buf_flags; /* copy of buf_flags */

```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+ __u64 hdr_reserved[10]; /* for future use */
+};
+
+/*
+ * Entry header in the sampling buffer. The header is directly followed
+ * with the values of the PMD registers of interest saved in increasing
+ * index order: PMD4, PMD5, and so on. How many PMDs are present depends
+ * on how the session was programmed.
+ *
+ * In the case where multiple counters overflow at the same time, multiple
+ * entries are written consecutively.
+ *
+ * last_reset_value member indicates the initial value of the overflowed PMD.
+ */
+struct pfm_dfl_smpl_entry {
+ __u32 pid; /* thread id (for NPTL, this is gettid()) */
+ __u16 ovfl_pmd; /* index of overflowed PMD for this sample */
+ __u16 reserved; /* for future use */
+ __u64 last_reset_val; /* initial value of overflowed PMD */
+ __u64 ip; /* where did the overflow interrupt happened */
+ __u64 tstamp; /* overflow timetamp */
+ __u16 cpu; /* cpu on which the overflow occurred */
+ __u16 set; /* event set active when overflow occurred */
+ __u32 tgid; /* thread group id (for NPTL, this is getpid())*/
+};
+
+#define PFM_DFL_SMPL_VERSION_MAJ 1U
+#define PFM_DFL_SMPL_VERSION_MIN 0U
+#define PFM_DFL_SMPL_VERSION (((PFM_DFL_SMPL_VERSION_MAJ&0xffff)<<16)\|
+ (PFM_DFL_SMPL_VERSION_MIN & 0xffff))
+
+#endif /* __PERFMON_DFL_SMPL_H__ */
--- linux-2.6.17-rc4.orig/include/linux/perfmon_fmt.h 1969-12-31 16:00:00.000000000 -0800
+++ linux-2.6.17-rc4/include/linux/perfmon_fmt.h 2006-05-12 03:18:52.000000000 -0700
@@ -0,0 +1,74 @@
+/*
+ * Copyright (c) 2001-2006 Hewlett-Packard Development Company, L.P.
+ * Contributed by Stephane Eranian <eranian@xxxxxxxxxxx>
+ *
+ * Interface for custom sampling buffer format modules
+ */
+#ifndef __PERFMON_FMT_H__
+#define __PERFMON_FMT_H__ 1
+
+#include <linux/kobject.h>
+
+struct pfm_ovfl_arg {
+ u16 ovfl_pmd; /* index of overflowed PMD */
+ u16 active_set; /* set active at the time of the overflow */
+ pfm_flags_t ovfl_ctrl; /* control flags */
+ u64 pmd_last_reset; /* last reset value of overflowed PMD */
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+ u64 smpl_pmds_values[PFM_MAX_PMDS]; /* values of other PMDs */
+ u64 pmd_eventid; /* eventid associated with PMD */
+ u16 num_smpl_pmds; /* number of PMDS in smpl_pmd_values */
+};
+
+/*
+ * ovfl_ctrl bitmask of flags
+ */
+#define PFM_OVFL_CTRL_NOTIFY 0x1 /* notify user */
+#define PFM_OVFL_CTRL_RESET 0x2 /* reset overflowed pmds */
+#define PFM_OVFL_CTRL_MASK 0x4 /* mask monitoring */
+
+
+typedef int (*fmt_validate_t)(u32 flags, u16 nmpds, void *arg);
+typedef int (*fmt_getsize_t)(u32 flags, void *arg, size_t *size);
+typedef int (*fmt_init_t)(struct pfm_context *ctx, void *buf, u32 flags, u16 nmpds, void *arg);
+typedef int (*fmt_restart_t)(int is_active, pfm_flags_t *ovfl_ctrl, void *buf);
+typedef int (*fmt_exit_t)(void *buf);
+typedef int (*fmt_handler_t)(void *buf, struct pfm_ovfl_arg *arg,
+ unsigned long ip, u64 stamp, void *data);
+
+struct pfm_smpl_fmt {
+ char *fmt_name; /* name of the format (required) */
+ pfm_uuid_t fmt_uuid; /* 128-bit unique id (required) */
+ size_t fmt_arg_size; /* size of fmt args for ctx create */
+ pfm_flags_t fmt_flags; /* format specific flags */
+
+ fmt_validate_t fmt_validate; /* validate context flags */
+ fmt_getsize_t fmt_getsize; /* get size for sampling buffer */
+ fmt_init_t fmt_init; /* initialize buffer area */
+ fmt_handler_t fmt_handler; /* overflow handler (required) */
+ fmt_restart_t fmt_restart; /* restart after notification */
+ fmt_exit_t fmt_exit; /* context termination */
+
+ struct list_head fmt_list; /* internal use only */
+
+ struct kobject kobj; /* sysfs internal use only */
+ struct module *owner; /* pointer to module owner */
+ u32 fmt_qdepth; /* Max notify queue depth (required) */
+};
+#define to_smpl_fmt(n) container_of(n, struct pfm_smpl_fmt, kobj)
+
+#define PFM_FMTFL_IS_BUILTIN 0x1 /* fmt is compiled in */
+/*
+ * we need to know whether the format is builtin or compiled
+ * as a module
+ */
+#ifdef MODULE
+#define PFM_FMT_BUILTIN_FLAG 0 /* not built as a module */
+#else
+#define PFM_FMT_BUILTIN_FLAG PFM_PMUFL_IS_BUILTIN /* built as a module */
```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+#endif
+
+int pfm_register_smpl_fmt(struct pfm_smpl_fmt *fmt);
+int pfm_unregister_smpl_fmt(pfm_uuid_t uuid);
+void pfm_builtin_fmt_sysfs_add(void);
+
+#endif /* __PERFMON_FMT_H__ */
--- linux-2.6.17-rc4.orig/include/linux/perfmon_kernel.h 1969-12-31 16:00:00.000000000 -0800
+++ linux-2.6.17-rc4/include/linux/perfmon_kernel.h 2006-05-12 03:18:52.000000000 -0700
@@ -0,0 +1,88 @@
+/*
+ * Copyright (c) 2006 Hewlett-Packard Development Company, L.P.
+ * Contributed by Stephane Eranian <eranian@xxxxxxxxxxx>
+ *
+ * Kernel hooks for perfmon
+ */
+#ifndef __PERFMON_KERNEL_H__
+#define __PERFMON_KERNEL_H__ 1
+
+#ifdef __KERNEL__
+
+#ifdef CONFIG_PERFMON
+
+void __pfm_exit_thread(struct task_struct *);
+void __pfm_copy_thread(struct task_struct *task);
+void __pfm_ctxswin(struct task_struct *task);
+void __pfm_ctxswout(struct task_struct *task);
+void __pfm_handle_work(void);
+void __pfm_handle_switch_timeout(void);
+void pfm_vector_init(void);
+
+
+static inline void pfm_exit_thread(struct task_struct *task)
+{
+ if (task->pfm_context)
+ __pfm_exit_thread(task);
+}
+
+static inline void pfm_handle_work(void)
+{
+ if (current->pfm_context)
+ __pfm_handle_work();
+}
+
+static inline void pfm_copy_thread(struct task_struct *task,
+ struct pt_regs *regs)
+{
+ /*
+ * task+regs are child state
+ */
+ if (task->pfm_context)
```

```

+__pfm_copy_thread(task);
+}
+
+static inline void pfm_ctxswin(struct task_struct *task)
+{
+ /*
+ * cannot check for pfm_context because
+ * not necessarily present in system-wide
+ */
+__pfm_ctxswin(task);
+}
+
+static inline void pfm_ctxswout(struct task_struct *task)
+{
+ /*
+ * cannot check for pfm_context because
+ * not necessarily present in system-wide
+ */
+__pfm_ctxswout(task);
+}
+
+static inline void pfm_handle_switch_timeout(void)
+{
+ unsigned long info;
+ info = __get_cpu_var(pfm_syst_info);
+ if (info & PFM_CPUINFO_TIME_SWITCH)
+ __pfm_handle_switch_timeout();
+}
+
+##else /* !CONFIG_PERFMON */
+
+
+##define pfm_exit_thread(_t) do { } while (0)
+##define pfm_handle_work() do { } while (0)
+##define pfm_copy_thread(_t,_r) do { } while (0)
+##define pfm_ctxswin(_t) do { } while (0)
+##define pfm_ctxswout(_t) do { } while (0)
+##define pfm_handle_switch_timeout() do { } while (0)
+##define pfm_vector_init() do { } while (0)
+##define pfm_release_dbregs(_t) do { } while (0)
+##define pfm_use_dbregs(_t) (0)
+
+##endif /* CONFIG_PERFMON */
+
+##endif /* __KERNEL__ */
+
+##endif /* __PERFMON_KERNEL_H__ */
--- linux-2.6.17-rc4.orig/include/linux/perfmon_pmu.h 1969-12-31 16:00:00.000000000 -0800
+++ linux-2.6.17-rc4/include/linux/perfmon_pmu.h 2006-05-12 03:18:52.000000000 -0700
@@ -0,0 +1,134 @@
+/*

```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+ * Copyright (c) 2006 Hewlett-Packard Development Company, L.P.
+ * Contributed by Stephane Eranian <eranian@xxxxxxxxxx>
+ *
+ * Interface for PMU description modules
+ */
+#ifndef __PERFMON_PMU_H__
+#define __PERFMON_PMU_H__ 1
+
+/*
+ * generic information about a PMC or PMD register
+ */
+struct pfm_reg_desc {
+ u16 type; /* role of the register */
+ char *desc; /* HW register description string */
+ u64 default_value; /* power-on default value (quiescent) */
+ u64 reserved_mask; /* reserved bits: 0 means reserved */
+ u64 no_emul64_mask; /* bits to clear for PFM_REGFL_NO_EMUL64 */
+};
+
+/*
+ * type of a PMU register (16-bit bitmask) for use with pfm_reg_desc.type
+ */
+#define PFM_REG_NA 0x00 /* not avail. (not impl.,no access) must be 0 */
+#define PFM_REG_I 0x01 /* implemented */
+#define PFM_REG_WC 0x02 /* has write_checker */
+#define PFM_REG_C64 0x04 /* PMD: 64-bit virtualization */
+#define PFM_REG_RO 0x08 /* PMD: read-only (writes ignored) */
+#define PFM_REG_V 0x10 /* PMD: virtual reg (provided by PMU description) */
+#define PFM_REG_NO64 0x100 /* PMC: supports REGFL_NOEMUL64 */
+
+/*
+ * define some shortcuts for common types
+ */
+#define PFM_REG_W (PFM_REG_WC|PFM_REG_I)
+#define PFM_REG_W64 (PFM_REG_WC|PFM_REG_NO64|PFM_REG_I)
+#define PFM_REG_C (PFM_REG_C64|PFM_REG_I)
+
+
+typedef int (*pfm_reg_check_t)(struct pfm_context *ctx,
+ struct pfm_event_set *set, u16 cnum, u32 flags, u64 *val);
+typedef u64 (*pfm_pmd_sread_t)(struct pfm_context *ctx, unsigned int cnum);
+typedef void (*pfm_pmd_swrite_t)(struct pfm_context *ctx, unsigned int cnum, u64 val);
+
+/*
+ * _pfm_pmu_config is for perfmon core/arch specific use only.
+ * PMU description modules must use struct pfm_pmu_config.
+ *
+ * The pfm_internal_pmu_config is shared between all CPUs. It is accessed for reading
+ * only, as such the cache lines will be shared (replicated) and we should not
+ * suffer any major penalty for having only one copy of the struct on large NUMA
+ * systems.
```

```

+ */
+struct _pfm_pmu_config {
+ u64 impl_pmcs[PFM_PMC_BV]; /* impl PMC */
+ u64 impl_pmds[PFM_PMD_BV]; /* impl PMD */
+ u64 impl_rw_pmds[PFM_PMD_BV]; /* impl RW PMD */
+ u64 cnt_pmds[PFM_PMD_BV]; /* impl counter */
+ u64 ovfl_mask; /* overflow mask */
+ u16 max_pmc; /* highest+1 impl PMC */
+ u16 max_pmd; /* highest+1 impl PMD */
+ u16 max_rw_pmd; /* highest+1 impl RW PMD */
+ u16 first_cnt_pmd; /* first counting PMD */
+ u16 max_cnt_pmd; /* highest+1 impl counter */
+ u16 num_pmcs; /* logical PMCS */
+ u16 num_pmds; /* logical PMDS */
+ u16 num_counters; /* PMC/PMD counter pairs */
+
+ char *pmu_name; /* PMU family name */
+ char *version; /* config module version number */
+ int counter_width; /* width of hardware counter */
+
+ struct pfm_reg_desc pmc_desc[PFM_MAX_PMCS+1]; /* PMC register descriptions */
+ struct pfm_reg_desc pmd_desc[PFM_MAX_PMDS+1]; /* PMD register descriptions */
+
+ pfm_reg_check_t pmc_write_check; /* PMC write checker callback */
+
+ pfm_pmd_sread_t pmd_sread; /* PMD model specific read */
+ pfm_pmd_swrite_t pmd_swrite; /* PMD model specific write */
+
+ void *arch_info; /* arch-specific information */
+ u32 flags; /* set of flags */
+
+ struct module *owner; /* pointer to module struct */
+ struct kobject kobj; /* for internal use only */
+};
+#define to_pmu(n) container_of(n, struct _pfm_pmu_config, kobj)
+
+/*
+ * structure used by pmu description modules
+ *
+ * probe_pmu() routine return value:
+ * - 1 means recognized PMU
+ * - 0 means not recognized PMU
+ */
+struct pfm_pmu_config {
+ char *pmu_name; /* PMU family name */
+ char *version; /* config module version number */
+ int counter_width; /* width of hardware counter */
+ struct pfm_reg_desc *pmc_desc; /* PMC register descriptions */
+ struct pfm_reg_desc *pmd_desc; /* PMD register descriptions */
+ pfm_reg_check_t pmc_write_check; /* PMC write checker callback */
+ pfm_reg_check_t pmd_write_check; /* PMD write checker callback */

```

[PATCH 3/11] perfmon2 patch for review: new generic header files

```
+ pfm_reg_check_t pmd_read_check; /* PMD read checker callback */
+ pfm_pmd_sread_t pmd_sread; /* PMD model specific read */
+ pfm_pmd_swrite_t pmd_swrite; /* PMD model specific write */
+ int (*probe_pmu)(void); /* probe PMU routine */
+ u16 num_pmc_entries; /* number of entries in pmc_desc */
+ u16 num_pmd_entries; /* number of entries in pmd_desc */
+ void *arch_info; /* arch-specific information */
+ u32 flags; /* set of flags */
+ struct module *owner; /* pointer to module struct */
+};
+
+/*
+ * pfm_pmu_config flags
+ */
+#define PFM_PMUFL_IS_BUILTIN 0x1 /* pmu config is compiled in */
+
+/*
+ * we need to know whether the PMU description is builtin or compiled
+ * as a module
+ */
+#ifdef MODULE
+#define PFM_PMU_BUILTIN_FLAG 0 /* not built as a module */
+#else
+#define PFM_PMU_BUILTIN_FLAG PFM_PMUFL_IS_BUILTIN /* built as a module */
+#endif
+
+int pfm_register_pmu_config(struct pfm_pmu_config *cfg);
+void pfm_unregister_pmu_config(struct pfm_pmu_config *cfg);
+u64 pfm_pmu_sread(struct pfm_context *ctx, unsigned int cnum);
+
+#endif /* __PERFMON_PMU_H__ */
```

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