```
/*
 * IBM Dynamic Probes
 * Copyright (c) International Business Machines Corp., 2000
 *
 * 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.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111−1307, USA.
 */
#include <linux/dprobes.h>
#include <linux/hook.h>

void dprobes_hooks_code_start(void) { }

USE_HOOK(SYS_INIT_MODULE);
USE_HOOK(SYS_INIT_MODULE_ERROR);
USE_HOOK(FREE_MODULE);

static int insmod_exit(hook_rec_t * hookrec, struct module * mod)
{
        dp_insmod(mod);
        return HOOK_CONTINUE;
}

static int remmod_exit(hook_rec_t * hookrec, struct module * mod)
{
        dp_remmod(mod);
        return HOOK_CONTINUE;
}

int initialise_dprobes_hooks()
{
        int retval;
        static int done = 0;
        if (done)
                return 0;
        done = 1;

        if ((retval = hook_initialise(SYS_INIT_MODULE)))
                goto err;
        if ((retval = hook_initialise(SYS_INIT_MODULE_ERROR)))
                goto err1;
        if ((retval = hook_initialise(FREE_MODULE)))
                goto err2;
        if ((retval = initialise_asm_dprobes_hooks()))
                goto err3;
        return 0;

err3:   hook_terminate(FREE_MODULE, 0);
err2:   hook_terminate(SYS_INIT_MODULE_ERROR, 0);
err1:   hook_terminate(SYS_INIT_MODULE, 0);
err:    return retval;
}

int terminate_dprobes_hooks()
{
        hook_terminate(SYS_INIT_MODULE, 0);
        hook_terminate(SYS_INIT_MODULE_ERROR, 0);
        hook_terminate(FREE_MODULE, 0);
        terminate_asm_dprobes_hooks();
        return 0;
}

hook_rec_t insmod_hook_rec;
hook_rec_t insmod_err_hook_rec;
hook_rec_t remmod_hook_rec;

int register_dprobes_hooks()
{
        int retval;
        static int done = 0;
        if (done)
                return 0;
        done = 1;

        insmod_hook_rec.hook_exit = insmod_exit;
        insmod_hook_rec.hook_exit_name = "dp_insmod_exit";
        if ((retval = hook_exit_register(SYS_INIT_MODULE, &insmod_hook_rec, 0)))
                goto err;

        insmod_err_hook_rec.hook_exit = remmod_exit;
```

```
91          insmod_hook_rec.hook_exit_name = "dp_remmod_exit";
92          if ((retval = hook_exit_register(SYS_INIT_MODULE_ERROR, &insmod_err_hook_rec, 0)))
93                  goto err1;
94
95          remmod_hook_rec.hook_exit = remmod_exit;
96          insmod_hook_rec.hook_exit_name = "dp_remmod_exit";
97          if ((retval = hook_exit_register(FREE_MODULE, &remmod_hook_rec, 0)))
98                  goto err2;
99
100         if ((retval = register_asm_dprobes_hooks()))
101                 goto err3;
102
103         return 0;
104
105 err3:   hook_exit_deregister(&remmod_hook_rec);
106 err2:   hook_exit_deregister(&insmod_err_hook_rec);
107 err1:   hook_exit_deregister(&insmod_hook_rec);
108 err:    return retval;
109 }
110
111 int deregister_dprobes_hooks()
112 {
113         hook_exit_deregister(&insmod_hook_rec);
114         hook_exit_deregister(&insmod_err_hook_rec);
115         hook_exit_deregister(&remmod_hook_rec);
116         deregister_asm_dprobes_hooks();
117         return 0;
118 }
119
120 int arm_all_dprobes_hooks()
121 {
122         int retval;
123
124         if ((retval = hook_exit_arm(&insmod_hook_rec)))
125                 goto err;
126         if ((retval = hook_exit_arm(&insmod_err_hook_rec)))
127                 goto err1;
128         if ((retval = hook_exit_arm(&remmod_hook_rec)))
129                 goto err2;
130         if ((retval = arm_all_asm_dprobes_hooks()))
131                 goto err3;
132         return 0;
133
134 err3:   hook_exit_disarm(&remmod_hook_rec);
135 err2:   hook_exit_disarm(&insmod_err_hook_rec);
136 err1:   hook_exit_disarm(&insmod_hook_rec);
137 err:    return retval;
138 }
139
140 int disarm_all_dprobes_hooks()
141 {
142         hook_exit_disarm(&remmod_hook_rec);
143         hook_exit_disarm(&insmod_err_hook_rec);
144         hook_exit_disarm(&insmod_hook_rec);
145         disarm_all_asm_dprobes_hooks();
146         return 0;
147 }
148
149 void dprobes_hooks_code_end(void) { }
```

```
/*
 * IBM Dynamic Probes
 * Copyright (c) International Business Machines Corp., 2000
 *
 * 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.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111−1307, USA.
 */

#include <linux/dprobes.h>
#include <linux/hook.h>

void dprobes_hooks_asm_code_start(void) { }

USE_HOOK(DO_TRAP);
USE_HOOK(DO_GPF);
USE_HOOK(DO_INT3);
USE_HOOK(DO_DEBUG);
USE_HOOK(MATH_STATE_RESTORE);
USE_HOOK(DO_PAGE_FAULT);
USE_HOOK(DPROBES_CALLK);

static int do_trap_exit(hook_rec_t * hookrec, struct pt_regs * regs)
{
        dp_handle_fault(regs);
        return HOOK_CONTINUE;
}

static int do_gpf_exit(hook_rec_t * hookrec, struct pt_regs * regs)
{
        if (dp_gpf(regs))
                return HOOK_RETURN;
        return HOOK_CONTINUE;
}

static int do_int3_exit(hook_rec_t * hookrec, int *rc, struct pt_regs * regs)
{
        if ((*rc = dp_trap(regs, DP_PROBE_BREAKPOINT, 0, regs->eip−1)))
                return HOOK_RETURN;
        return HOOK_CONTINUE;
}

static int do_debug_exit(hook_rec_t * hookrec, int *rc,
                struct pt_regs * regs, unsigned long condition)
{
        if ((*rc = dp_do_debug(regs, condition)))
                return HOOK_RETURN;
        return HOOK_CONTINUE;
}

static int do_page_fault_exit(hook_rec_t * hookrec, struct pt_regs * regs)
{
        if (dp_pf(regs))
                return HOOK_RETURN;
        return HOOK_CONTINUE;
}

int initialise_asm_dprobes_hooks(void)
{
        int retval;

        if ((retval = hook_initialise(DO_TRAP)))
                goto err;
        if ((retval = hook_initialise(DO_GPF)))
                goto err1;
        if ((retval = hook_initialise(DO_INT3)))
                goto err2;
        if ((retval = hook_initialise(DO_DEBUG)))
                goto err3;
        if ((retval = hook_initialise(MATH_STATE_RESTORE)))
                goto err4;
        if ((retval = hook_initialise(DO_PAGE_FAULT)))
                goto err5;
        if ((retval = hook_initialise(DPROBES_CALLK)))
                goto err6;
        return 0;

err6:   hook_terminate(DPROBES_CALLK, 0);
err5:   hook_terminate(MATH_STATE_RESTORE, 0);
err4:   hook_terminate(DO_DEBUG, 0);
```

```
91  err3:   hook_terminate(DO_INT3, 0);
92  err2:   hook_terminate(DO_GPF, 0);
93  err1:   hook_terminate(DO_TRAP, 0);
94  err:    return retval;
95  }
96
97  int terminate_asm_dprobes_hooks(void)
98  {
99          hook_terminate(DPROBES_CALLK, 0);
100         hook_terminate(DO_PAGE_FAULT, 0);
101         hook_terminate(MATH_STATE_RESTORE, 0);
102         hook_terminate(DO_DEBUG, 0);
103         hook_terminate(DO_INT3, 0);
104         hook_terminate(DO_GPF, 0);
105         hook_terminate(DO_TRAP, 0);
106         return 0;
107 }
108 hook_rec_t do_trap_hook_rec;
109 hook_rec_t do_gpf_hook_rec;
110 hook_rec_t do_int3_hook_rec;
111 hook_rec_t do_debug_hook_rec;
112 hook_rec_t math_state_hook_rec;
113 hook_rec_t do_page_fault_hook_rec;
114
115 int register_asm_dprobes_hooks(void)
116 {
117         int retval;
118
119         do_trap_hook_rec.hook_exit = do_trap_exit;
120         do_trap_hook_rec.hook_exit_name = "dp_do_trap_exit";
121         if ((retval = hook_exit_register(DO_TRAP, &do_trap_hook_rec, 0)))
122                 goto err;
123
124         do_gpf_hook_rec.hook_exit = do_gpf_exit;
125         do_gpf_hook_rec.hook_exit_name = "dp_do_gpf_exit";
126         if ((retval = hook_exit_register(DO_GPF, &do_gpf_hook_rec, 0)))
127                 goto err1;
128
129         do_int3_hook_rec.hook_exit = do_int3_exit;
130         do_int3_hook_rec.hook_exit_name = "dp_do_int3_exit";
131         if ((retval = hook_exit_register(DO_INT3, &do_int3_hook_rec, 0)))
132                 goto err2;
133
134         do_debug_hook_rec.hook_exit = do_debug_exit;
135         do_debug_hook_rec.hook_exit_name = "dp_do_debug_exit";
136         if ((retval = hook_exit_register(DO_DEBUG, &do_debug_hook_rec, 0)))
137                 goto err3;
138
139         math_state_hook_rec.hook_exit = do_trap_exit;
140         math_state_hook_rec.hook_exit_name = "dp_do_trap_exit";
141         if ((retval = hook_exit_register(MATH_STATE_RESTORE, &math_state_hook_rec, 0)))
142                 goto err4;
143
144         do_page_fault_hook_rec.hook_exit = do_page_fault_exit;
145         do_page_fault_hook_rec.hook_exit_name = "dp_do_page_fault_exit";
146         if ((retval = hook_exit_register(DO_PAGE_FAULT, &do_page_fault_hook_rec, 0)))
147                 goto err5;
148         return 0;
149
150 err5:   hook_exit_deregister(&math_state_hook_rec);
151 err4:   hook_exit_deregister(&do_debug_hook_rec);
152 err3:   hook_exit_deregister(&do_int3_hook_rec);
153 err2:   hook_exit_deregister(&do_gpf_hook_rec);
154 err1:   hook_exit_deregister(&do_trap_hook_rec);
155 err:    return retval;
156
157 }
158
159 int deregister_asm_dprobes_hooks(void)
160 {
161         hook_exit_deregister(&do_page_fault_hook_rec);
162         hook_exit_deregister(&math_state_hook_rec);
163         hook_exit_deregister(&do_debug_hook_rec);
164         hook_exit_deregister(&do_int3_hook_rec);
165         hook_exit_deregister(&do_gpf_hook_rec);
166         hook_exit_deregister(&do_trap_hook_rec);
167         return 0;
168 }
169
170 int arm_all_asm_dprobes_hooks(void)
171 {
172         int retval;
173
174         if ((retval = hook_exit_arm(&do_trap_hook_rec)))
175                 goto err;
176         if ((retval = hook_exit_arm(&do_gpf_hook_rec)))
177                 goto err1;
178         if ((retval = hook_exit_arm(&do_int3_hook_rec)))
179                 goto err2;
180         if ((retval = hook_exit_arm(&do_debug_hook_rec)))
```

```
181                 goto err3;
182         if ((retval = hook_exit_arm(&math_state_hook_rec)))
183                 goto err4;
184         if ((retval = hook_exit_arm(&do_page_fault_hook_rec)))
185                 goto err5;
186         return 0;
187
188 err5:   hook_exit_disarm(&math_state_hook_rec);
189 err4:   hook_exit_disarm(&do_debug_hook_rec);
190 err3:   hook_exit_disarm(&do_int3_hook_rec);
191 err2:   hook_exit_disarm(&do_gpf_hook_rec);
192 err1:   hook_exit_disarm(&do_trap_hook_rec);
193 err:    return retval;
194 }
195
196 int disarm_all_asm_dprobes_hooks(void)
197 {
198         hook_exit_disarm(&do_page_fault_hook_rec);
199         hook_exit_disarm(&math_state_hook_rec);
200         hook_exit_disarm(&do_debug_hook_rec);
201         hook_exit_disarm(&do_int3_hook_rec);
202         hook_exit_disarm(&do_gpf_hook_rec);
203         hook_exit_disarm(&do_trap_hook_rec);
204         return 0;
205 }
206
207 void dprobes_hooks_asm_code_end(void) { }
```

```
/*
 * IBM Dynamic Probes
 * Copyright (c) International Business Machines Corp., 2000
 *
 * 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.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111−1307, USA.
 */

/*
 * This is a special file, part of the dprobes interpreter that contains
 * architecture−specific functions. It is not compiled as a seperate unit.
 * It is #included into the arch−independent dprobes_in.c at a specific
 * location.
 */
#ifdef CONFIG_DUMP
#include <linux/dump.h>
#endif
#ifdef CONFIG_KDB
#include <linux/kdb.h>
#endif
#include <asm/desc.h>

/*
 * Convert segmented address (selector : offset) to flat linear address.
 */
static void seg2lin(void)
{
        void *faddr;
        unsigned long off = rpnpop();
        unsigned long sel = rpnpop();

        if (dp_seg_to_flat(sel, off, &faddr)) {
                gen_ex(EX_SEG_FAULT, sel, 0);
                return;
        }
        rpnpush((unsigned long) faddr);
}

#ifdef CONFIG_KDB
/*
 * Temporarily restore the orignal opcode befor calling the kdb
 * so that kdb shows correct disassembly.
 */
static inline void restore_opcode(void)
{
        if (dprobes.rec−>point.probe & DP_PROBE_BREAKPOINT) {
                dprobes.reset_addr = dprobes.probe_addr;
                *((byte_t *)dprobes.reset_addr) = dprobes.opcode;
                flush_page_to_ram(dprobes.reset_addr);
        }
}

/*
 * After kdb terminates, restore the breakpoint.
 */
static inline void restore_int3(void)
{
        if (dprobes.rec−>point.probe & DP_PROBE_BREAKPOINT) {
                dprobes.reset_addr = dprobes.probe_addr;
                *((byte_t *)dprobes.reset_addr) = DP_INSTR_BREAKPOINT;
                flush_page_to_ram(dprobes.reset_addr);
        }
}

void call_kdb(void)
{
        unsigned long eip = dprobes.regs−>eip − 1;
        if (dprobes.status & DP_KERNEL_PROBE) {
                if (dprobes.rec−>point.probe & DP_PROBE_BREAKPOINT)
                        dprobes.regs−>eip−−;
                restore_opcode();
                kdb(KDB_REASON_CALL, 0, dprobes.regs);
                restore_int3();
                if (eip == dprobes.regs−>eip)
                        dprobes.regs−>eip++;
        }
}
#else

```

```
91  void call_kdb(void)
92  {
93          printk("Dprobes: Kernel Debugger is not present.\n");
94          return;
95  }
96  #endif
97  /*
98   * Exiting the interpreter through the registered facility n.
99   */
100 static void dp_exit_n(void)
101 {
102         u8 facility = get_u8_oprnd();
103         switch(facility) {
104         case DP_EXIT_TO_SGI_KDB:
105                 call_kdb();
106                 break;
107
108         case DP_EXIT_TO_SGI_VMDUMP:
109 #ifdef CONFIG_DUMP
110                 if (dprobes.status & DP_KERNEL_PROBE)
111                         dump("Dumping due to dprobes", dprobes.regs);
112 #else
113                 printk("dprobes: Crash dump facility is not present.\n");
114 #endif
115                 break;
116         case DP_EXIT_TO_CORE_DUMP:
117                 if (dprobes.status & DP_USER_PROBE) {
118                         if (do_coredump(SIGINT, dprobes.regs)) {
119                                 current->mm->dumpable = 1;
120                                 printk("dprobes(%d,%d) process %s dumped core\n", dprobes.major, dprobes.minor, current->c
    omm);
121                         }
122                         else
123                                 printk("dprobes(%d,%d) exit to core dump failed\n", dprobes.major, dprobes.minor);
124                 }
125                 break;
126         default: gen_ex(EX_INVALID_OPERAND, 4, facility); return;
127
128         }
129         dprobes.status &= ~DP_STATUS_INTERPRETER;
130 }
131
132 /*
133  * Interpreter's version of copy_xxx_user functions. These need to save
134  * and restore cr2 contents to be able to correctly handle probes in
135  * page fault handler.
136  */
137 static int dp_intr_copy_from_user(void *to, void *from, int len)
138 {
139         unsigned long address;
140         int retval;
141         __asm__("movl %%cr2,%0":"=r" (address)); /* save cr2 contents */
142         retval = __copy_from_user(to, from, len);
143         __asm__("movl %0,%%cr2": :"r" (address)); /* restore cr2 contents */
144         return retval;
145 }
146
147 static int dp_intr_copy_to_user(void *to, void *from, int len)
148 {
149         unsigned long address;
150         int retval;
151         __asm__("movl %%cr2,%0":"=r" (address)); /* save cr2 contents */
152         retval = __copy_to_user(to, from, len);
153         __asm__("movl %0,%%cr2": :"r" (address)); /* restore cr2 contents */
154         return retval;
155 }
156
157 #define BITS_IN_UL      32
158
159 #define PROPAGATE_BIT(name, OPERATOR) \
160 static void name(void) \
161 { \
162         unsigned long index, num, pos, val; \
163         index = rpnpop(); \
164         if (!index || index > BITS_IN_UL) { \
165                 gen_ex(EX_INVALID_OPERAND, 3, index); \
166                 return; \
167         } \
168         num = rpnpop(); \
169         pos = 1UL << (index - 1); \
170         val = num & pos; \
171         for (; pos; pos OPERATOR 1, val OPERATOR 1) \
172                 num &= ~pos, num |= val; \
173         rpnpush(num); \
174 }
175
176 PROPAGATE_BIT(pbl, <<=)
177 PROPAGATE_BIT(pbr, >>=)
178
179 #define PROPAGATE_BIT_IMM(name, OPERATOR) \
```

```
180 static void name(void) \
181 { \
182         unsigned long index, num, pos, val; \
183         index = (unsigned long)get_u8_oprnd(); \
184         if (!index || index > BITS_IN_UL) { \
185                 dprobes.status &= ~DP_STATUS_INTERPRETER; \
186                 return; \
187         } \
188         num = rpnpop(); \
189         pos = 1UL << (index - 1); \
190         val = num & pos; \
191         for (; pos; pos OPERATOR 1, val OPERATOR 1) \
192                 num &= ~pos, num |= val; \
193         rpnpush(num); \
194 }
195
196 PROPAGATE_BIT_IMM(pbl_i, <<=)
197 PROPAGATE_BIT_IMM(pbr_i, >>=)
198
199 static void pzl(void)
200 {
201         unsigned long index, num, pos;
202         index = rpnpop();
203         if (!index || index > BITS_IN_UL) {
204                 gen_ex(EX_INVALID_OPERAND, 3, index); \
205                 return;
206         }
207         num = rpnpop();
208         pos = 1UL << index;
209         for (; pos; pos <<= 1)
210                 num &= ~pos;
211         rpnpush(num);
212 }
213
214 static void pzl_i(void)
215 {
216         unsigned long index, num, pos;
217         index = (unsigned long)get_u8_oprnd();
218         if (!index || index > BITS_IN_UL) {
219                 gen_ex(EX_INVALID_OPERAND, 3, index);
220                 return;
221         }
222         num = rpnpop();
223         pos = 1UL << index;
224         for (; pos; pos <<= 1)
225                 num &= ~pos;
226         rpnpush(num);
227 }
228
229 static void pzr(void)
230 {
231         unsigned long index, num, pos;
232         index = rpnpop();
233         if (!index || index > BITS_IN_UL) {
234                 gen_ex(EX_INVALID_OPERAND, 3, index);
235                 return;
236         }
237         if (index == 1)
238                 return;
239         num = rpnpop();
240         pos = 1UL << (index - 2);
241         for (; pos; pos >>= 1)
242                 num &= ~pos;
243         rpnpush(num);
244 }
245
246 static void pzr_i(void)
247 {
248         unsigned long index, num, pos;
249         index = (unsigned long)get_u8_oprnd();
250         if (!index || index > BITS_IN_UL) {
251                 gen_ex(EX_INVALID_OPERAND, 3, index);
252                 return;
253         }
254         if (index == 1)
255                 return;
256         num = rpnpop();
257         pos = 1UL << (index - 2);
258         for (; pos; pos >>= 1)
259                 num &= ~pos;
260         rpnpush(num);
261 }
262
263 static void ror_i(void)
264 {
265         byte_t count = get_u8_oprnd();
266         __asm__ __volatile__("rorl %%cl, %0"
267                              :"=m"(dprobes.rpn_stack[dprobes.rpn_tos])
268                              :"c"((unsigned long) count));
269 }
```

```
270
271 static void rol_i(void)
272 {
273         byte_t count = get_u8_oprnd();
274         __asm__ __volatile__("roll %%cl, %0"
275                              :"=m"(dprobes.rpn_stack[dprobes.rpn_tos])
276                              :"c"((unsigned long) count));
277 }
278
279 static void ror(void)
280 {
281         unsigned long oprnd = rpnpop();
282         unsigned long count = rpnpop();
283         __asm__ __volatile__("rorl %%cl, %0" : "=m"(oprnd)
284                              :"c"(count) );
285         rpnpush(oprnd);
286 }
287
288 static void rol(void)
289 {
290         unsigned long oprnd = rpnpop();
291         unsigned long count = rpnpop();
292         __asm__ __volatile__("roll %%cl, %0" : "=m"(oprnd)
293                              :"c"(count));
294         rpnpush(oprnd);
295 }
296
297 /*
298  * IO group
299  */
300 static void push_io_u8(void)
301 {
302         rpnpush(inb(rpnpop()));
303 }
304 static void push_io_u16(void)
305 {
306         rpnpush(inw(rpnpop()));
307 }
308
309 static void push_io_u32(void)
310 {
311         rpnpush(inl(rpnpop()));
312 }
313
314 static void pop_io_u8(void)
315 {
316         u8 b = (u8)rpnpop();
317         outb(b, rpnpop());
318 }
319
320 static void pop_io_u16(void)
321 {
322         u16 w = (u16)rpnpop();
323         outw(w, rpnpop());
324 }
325
326 static void pop_io_u32(void)
327 {
328         u32 d = (u32)rpnpop();
329         outl(d, rpnpop());
330 }
331
332 /*
333  * Register push and pop, current and user registers.
334  */
335
336 /*
337  * y -- allowed and implemented, n -- not allowed and not implemented.
338  *
339  * Index        Register        push u  push r  pop u   pop r
340  *
341  * 0            cs              y       y       n       n
342  * 1            ds              y       y       y       y
343  * 2            es              y       y       y       y
344  * 3            fs              y       y       y       y
345  * 4            gs              y       y       y       y
346  * 5            ss              y       y       n       n
347  * 6            eax             y       y       y       y
348  * 7            ebx             y       y       y       y
349  * 8            ecx             y       y       y       y
350  * 9            edx             y       y       y       y
351  * a            edi             y       y       y       y
352  * b            esi             y       y       y       y
353  * c            eflags          y       y       n       n
354  * d            eip             y       y       n       n
355  * e            esp             y       y       n       n
356  * f            ebp             y       y       y       y
357  * 20           tr              n       y       n       n
358  * 21           ldtr            n       y       n       n
359  * 22           gdtr            n       y       n       n
```

```
 * 23            idtr            n       y       n       n
 * 24            cr0             n       y       n       n
 * 25            cr1                     RESERVED
 * 26            cr2             n       y       n       n
 * 27            cr3             n       y       n       n
 * 28            cr4             n       y       n       n
 * 29            cr5                     RESERVED
 * 2a            cr6                     RESERVED
 * 2b            cr7                     RESERVED
 * 2c            dr0             n       y       n       n
 * 2d            dr1             n       y       n       n
 * 2e            dr2             n       y       n       n
 * 2f            dr3             n       y       n       n
 * 30            dr4                     RESERVED
 * 31            dr5                     RESERVED
 * 32            dr6             n       y       n       n
 * 33            dr7             n       y       n       n
 * 34            tr0                     RESERVED
 * 35            tr1                     RESERVED
 * 36            tr2                     RESERVED
 * 37            tr3                     RESERVED
 * 38            tr4                     RESERVED
 * 39            tr5                     RESERVED
 * 3a            tr6                     RESERVED
 * 3b            tr7                     RESERVED
 * 3c            cpuid           y       y       n       n
 * 3d            msr             y       y       n       n
 * 3e            fr0             y       n       n       n
 * 3f            fr1             y       n       n       n
 * 40            fr2             y       n       n       n
 * 41            fr3             y       n       n       n
 * 42            fr4             y       n       n       n
 * 43            fr5             y       n       n       n
 * 44            fr6             y       n       n       n
 * 45            fr7             y       n       n       n
 * 46            fcw             y       n       n       n
 * 47            fsw             y       n       n       n
 * 48            ftw             y       n       n       n
 * 49            fip             y       n       n       n
 * 4a            fcs             y       n       n       n
 * 4b            fdp             y       n       n       n
 * 4c            fds             y       n       n       n
 * 4d            xmm0            y       n       n       n
 * 4e            xmm1            y       n       n       n
 * 4f            xmm2            y       n       n       n
 * 50            xmm3            y       n       n       n
 * 51            xmm4            y       n       n       n
 * 52            xmm5            y       n       n       n
 * 53            xmm6            y       n       n       n
 * 54            xmm7            y       n       n       n
 * 55            mxcsr           y       n       n       n
 */

static void dp_save_fpu(void)
{
        dprobes.status &= ~DP_STATUS_FIRSTFPU;
        if (HAVE_HWFP) {
                if (!current->used_math) {
                        memset(&dprobes.fpu_save_area,0, sizeof(dprobes.fpu_save_area));
                        return;
                }

                if ((read_cr0()) & CR0_TS) {
                        dprobes.fpu_save_area = current->thread.i387;
                }
                else {
                        if (cpu_has_fxsr) {
                                __asm__ __volatile__ (
                                        "fxsave %0\n"
                                        "fxrstor %0" :"=m" (dprobes.fpu_save_area));
                        } else {
                                __asm__ __volatile__ (
                                        "fsave %0\n"
                                        "frstor %0" :"=m" (dprobes.fpu_save_area));

                        }
                }
        }
        else
                memset(&dprobes.fpu_save_area,0, sizeof(dprobes.fpu_save_area));

}

/*
 * Define Floating Point Control Registers' push functions.
 */

#define DEFINE_PUSHFP(REG) \
static void pushfp_##REG(void) \
{ \
```

```
450         if (dprobes.status & DP_STATUS_FIRSTFPU) \
451                 dp_save_fpu(); \
452         if (cpu_has_fxsr) \
453                 rpnpush(dprobes.fpu_save_area.fxsave.REG); \
454         else \
455                 rpnpush(dprobes.fpu_save_area.fsave.REG); \
456 }
457
458 DEFINE_PUSHFP(cwd)
459 DEFINE_PUSHFP(swd)
460 DEFINE_PUSHFP(twd)
461 DEFINE_PUSHFP(fip)
462 DEFINE_PUSHFP(fcs)
463 DEFINE_PUSHFP(foo)
464 DEFINE_PUSHFP(fos)
465
466 static void pushfp_mxcsr(void)
467 {
468         if (dprobes.status & DP_STATUS_FIRSTFPU)
469                 dp_save_fpu();
470         if (cpu_has_xmm)
471                 rpnpush(dprobes.fpu_save_area.fxsave.mxcsr);
472         else
473                 rpnpush(0); /* could we return 0x1f80 ? */
474 }
475
476 /*
477  * Floating Point Data Registers' push function.
478  */
479 static void pushfp_st(int offset)
480 {
481         unsigned long st_0, st_4, st_8;
482         if (dprobes.status & DP_STATUS_FIRSTFPU)
483                 dp_save_fpu();
484         if (cpu_has_fxsr) {
485                 offset = (offset - 0x3e) * 16;
486                 st_0 = *(unsigned long *) ((unsigned char *)
487                         (dprobes.fpu_save_area.fxsave.st_space) + offset);
488                 st_4 = *(unsigned long *) ((unsigned char *)
489                         (dprobes.fpu_save_area.fxsave.st_space) + offset + 4);
490                 st_8 = *(unsigned long *) ((unsigned char *)
491                         (dprobes.fpu_save_area.fxsave.st_space) + offset + 8);
492         }
493         else {
494                 offset = (offset - 0x3e) * 10;
495                 st_0 = *(unsigned long *) ((unsigned char *)
496                         (dprobes.fpu_save_area.fsave.st_space) + offset);
497                 st_4 = *(unsigned long *) ((unsigned char *)
498                         (dprobes.fpu_save_area.fsave.st_space) + offset + 4);
499                 st_8 = *(unsigned short *) ((unsigned char *)
500                         (dprobes.fpu_save_area.fsave.st_space) + offset + 8);
501         }
502         rpnpush(st_8);
503         rpnpush(st_4);
504         rpnpush(st_0);
505 }
506
507 static void push_xmm(int offset)
508 {
509         unsigned long xmm_0 = 0, xmm_4 = 0, xmm_8 = 0, xmm_12 = 0;
510         if (dprobes.status & DP_STATUS_FIRSTFPU)
511                 dp_save_fpu();
512         if (cpu_has_xmm) {
513                 offset = (offset - 0x4d) * 16;
514                 xmm_0 = *(unsigned long *) ((unsigned char *)
515                         (dprobes.fpu_save_area.fxsave.xmm_space) + offset);
516                 xmm_4 = *(unsigned long *) ((unsigned char *)
517                         (dprobes.fpu_save_area.fxsave.xmm_space) + offset + 4);
518                 xmm_8 = *(unsigned long *) ((unsigned char *)
519                         (dprobes.fpu_save_area.fxsave.xmm_space) + offset + 8);
520                 xmm_12 = *(unsigned long *) ((unsigned char *)
521                         (dprobes.fpu_save_area.fxsave.xmm_space) + offset + 12);
522         }
523         rpnpush(xmm_12);
524         rpnpush(xmm_8);
525         rpnpush(xmm_4);
526         rpnpush(xmm_0);
527 }
528
529 /*
530  * push r, ss.
531  */
532 static unsigned long getr_ss(void)
533 {
534         unsigned long reg;
535         if (dprobes.status & DP_KERNEL_PROBE)
536                 __asm__ __volatile__("movw %%ss, %%ax\n\t"
537                                      :"=a"(reg)
538                                     );
539         else
```

```
540                     reg = dprobes.regs->xss;
541             return reg;
542     }
543
544     /*
545      * push r, esp
546      */
547     static unsigned long getr_esp(void)
548     {
549             if (dprobes.status & DP_KERNEL_PROBE)
550                     return (unsigned long)&dprobes.regs->esp;
551             else
552                     return dprobes.regs->esp;
553     }
554
555     /*
556      * Push GDTR
557      */
558     static void push_gdtr(void)
559     {
560             struct Xgt_desc_struct gdtr;
561             __asm__ __volatile__("sgdt (%0)" : : "r"((unsigned long) &gdtr));
562
563             rpnpush(gdtr.address);
564             rpnpush((unsigned long) gdtr.size);
565     }
566
567     /*
568      * Push LDTR
569      */
570     static void push_ldtr(void)
571     {
572             unsigned long ldtr = 0;
573             __asm__ __volatile__("sldt %0" : "=r"(ldtr));
574             rpnpush(ldtr);
575     }
576
577     /*
578      * Push IDTR
579      */
580     static void push_idtr(void)
581     {
582             struct Xgt_desc_struct idtr;
583             __asm__ __volatile__("sidt (%0)" : : "r"((unsigned long) &idtr));
584
585             rpnpush(idtr.address);
586             rpnpush((unsigned long)idtr.size);
587     }
588
589     /*
590      * Push TR
591      */
592     static void push_tr(void)
593     {
594             unsigned long tr;
595             __asm__ __volatile__("str %%ax" : "=a"(tr));
596             rpnpush(tr);
597     }
598
599     /*
600      * Push cpuid.
601      */
602     static void push_cpuid(void)
603     {
604             int op, cpuid_eax = 0, cpuid_ebx = 0, cpuid_ecx = 0, cpuid_edx = 0;
605             op = (int) rpnpop();
606             if ((boot_cpu_data.x86 >= 5) && (op <= boot_cpu_data.cpuid_level))
607                     cpuid(op, &cpuid_eax, &cpuid_ebx, &cpuid_ecx, &cpuid_edx);
608             rpnpush(cpuid_edx);
609             rpnpush(cpuid_ecx);
610             rpnpush(cpuid_ebx);
611             rpnpush(cpuid_eax);
612     }
613
614     /*
615      * push msr.
616      */
617     static void push_msr(void)
618     {
619             unsigned long msr, val1 =0, val2 = 0;
620             msr = rpnpop();
621             if (boot_cpu_data.x86 >= 5)
622                     __asm__ __volatile__ (
623                             "0:             rdmsr\n"
624                             "1:\n"
625                             ".section .fixup,\"ax\"\n"
626                             "2:             jmp 1b\n"
627                             ".previous\n"
628                             ".section __ex_table,\"a\"\n"
629                             "       .align 4\n"
```

```
630                             "      .long 0b,2b\n"
631                             ".previous"
632                             :"=a"(val1), "=d"(val2)
633                             :"c"(msr));
634             rpnpush(val2);
635             rpnpush(val1);
636     }
637
638     /*
639      * Prototype for control registers'(CR0 TO CR4) and debug registers'(DR0 TO DR7)
640      * push functions.
641      */
642     #define DEFINE_PUSHCRDB(REG) \
643     static void push_##REG(void) \
644     { \
645             unsigned long reg; \
646             __asm__ __volatile__ ("movl %%" #REG ", %0" : "=r"(reg)); \
647             rpnpush(reg); \
648     }
649
650     /*
651      * Define control and debug registers' push functions.
652      */
653     DEFINE_PUSHCRDB(cr0)
654     DEFINE_PUSHCRDB(cr2)
655     DEFINE_PUSHCRDB(cr3)
656     DEFINE_PUSHCRDB(cr4)
657     DEFINE_PUSHCRDB(dr0)
658     DEFINE_PUSHCRDB(dr1)
659     DEFINE_PUSHCRDB(dr2)
660     DEFINE_PUSHCRDB(dr3)
661     DEFINE_PUSHCRDB(dr6)
662     DEFINE_PUSHCRDB(dr7)
663
664     /*
665      * Since fs and gs are not in pt_regs, these
666      * are directly taken from the processor.
667      */
668     static unsigned long read_fs(void)
669     {
670             unsigned long val;
671             __asm__ __volatile__("movw %%fs, %%ax" :"=a"(val));
672             return val;
673     }
674
675     static unsigned long read_gs(void)
676     {
677             unsigned long val;
678             __asm__ __volatile__("movw %%gs, %%ax" :"=a"(val));
679             return val;
680     }
681
682     static void write_fs(unsigned long val)
683     {
684             __asm__ __volatile__("movw %%ax, %%fs" : : "a"(val));
685     }
686
687     static void write_gs(unsigned long val)
688     {
689             __asm__ __volatile__("movw %%ax, %%gs" : :"a"(val));
690     }
691
692     /*
693      * Current register main push function.
694      */
695     static void pushr(void)
696     {
697             struct dprobes_struct *dp = &dprobes;
698             unsigned short index = get_u16_oprnd();
699             unsigned long val;
700             switch(index) {
701                     case DP_CS:     val = dp->regs->xcs; break;
702                     case DP_DS:     val = dp->regs->xds; break;
703                     case DP_ES:     val = dp->regs->xes; break;
704                     case DP_FS:     val = read_fs(); break;
705                     case DP_GS:     val = read_gs(); break;
706                     case DP_SS:     val = getr_ss(); break;
707                     case DP_EAX:    val = dp->regs->eax; break;
708                     case DP_EBX:    val = dp->regs->ebx; break;
709                     case DP_ECX:    val = dp->regs->ecx; break;
710                     case DP_EDX:    val = dp->regs->edx; break;
711                     case DP_EDI:    val = dp->regs->edi; break;
712                     case DP_ESI:    val = dp->regs->esi; break;
713                     case DP_EFLAGS: val = dp->regs->eflags; break;
714                     case DP_EIP:    val = dp->regs->eip; break;
715                     case DP_ESP:    val = getr_esp(); break;
716                     case DP_EBP:    val = dp->regs->ebp; break;
717
718                     /* special registers */
719                     case DP_TR:     push_tr(); return;
```

```
720                     case DP_LDTR:   push_ldtr(); return;
721                     case DP_GDTR:   push_gdtr(); return;
722                     case DP_IDTR:   push_idtr(); return;
723                     case DP_CR0:    push_cr0(); return;
724                     case DP_CR2:    push_cr2(); return;
725                     case DP_CR3:    push_cr3(); return;
726                     case DP_CR4:    push_cr4(); return;
727                     case DP_DR0:    push_dr0(); return;
728                     case DP_DR1:    push_dr1(); return;
729                     case DP_DR2:    push_dr2(); return;
730                     case DP_DR3:    push_dr3(); return;
731                     case DP_DR6:    push_dr6(); return;
732                     case DP_DR7:    push_dr7(); return;
733                     case DP_CPUID:  push_cpuid(); return;
734                     case DP_MSR:    push_msr(); return;
735                     default:        val = 0; break;
736             }
737             rpnpush(val);
738             return;
739     }
740
741     /*
742      * User register main push function.
743      */
744     static void pushu(void)
745     {
746             struct dprobes_struct *dp = &dprobes;
747             unsigned short index = get_u16_oprnd();
748             unsigned long val;
749             switch(index) {
750                     case DP_CS:     val = dp->uregs->xcs; break;
751                     case DP_DS:     val = dp->uregs->xds; break;
752                     case DP_ES:     val = dp->uregs->xes; break;
753                     case DP_FS:     val = read_fs(); break;
754                     case DP_GS:     val = read_gs(); break;
755                     case DP_SS:     val = dp->uregs->xss; break;
756                     case DP_EAX:    val = dp->uregs->eax; break;
757                     case DP_EBX:    val = dp->uregs->ebx; break;
758                     case DP_ECX:    val = dp->uregs->ecx; break;
759                     case DP_EDX:    val = dp->uregs->edx; break;
760                     case DP_EDI:    val = dp->uregs->edi; break;
761                     case DP_ESI:    val = dp->uregs->esi; break;
762                     case DP_EFLAGS: val = dp->uregs->eflags; break;
763                     case DP_EIP:    val = dp->uregs->eip; break;
764                     case DP_ESP:    val = dp->uregs->esp; break;
765                     case DP_EBP:    val = dp->uregs->ebp; break;
766                     case DP_CPUID:  push_cpuid(); return;
767                     case DP_MSR:    push_msr(); return;
768                     case DP_FR0:    pushfp_st(0x3e); return;
769                     case DP_FR1:    pushfp_st(0x3f); return;
770                     case DP_FR2:    pushfp_st(0x40); return;
771                     case DP_FR3:    pushfp_st(0x41); return;
772                     case DP_FR4:    pushfp_st(0x42); return;
773                     case DP_FR5:    pushfp_st(0x43); return;
774                     case DP_FR6:    pushfp_st(0x44); return;
775                     case DP_FR7:    pushfp_st(0x45); return;
776                     case DP_FCW:    pushfp_cwd(); return;
777                     case DP_FSW:    pushfp_swd(); return;
778                     case DP_FTW:    pushfp_twd(); return;
779                     case DP_FIP:    pushfp_fip(); return;
780                     case DP_FCS:    pushfp_fcs(); return;
781                     case DP_FDP:    pushfp_foo(); return;
782                     case DP_FDS:    pushfp_fos(); return;
783
784                     case DP_XMM0:   push_xmm(0x4d); return;
785                     case DP_XMM1:   push_xmm(0x4e); return;
786                     case DP_XMM2:   push_xmm(0x4f); return;
787                     case DP_XMM3:   push_xmm(0x50); return;
788                     case DP_XMM4:   push_xmm(0x51); return;
789                     case DP_XMM5:   push_xmm(0x52); return;
790                     case DP_XMM6:   push_xmm(0x53); return;
791                     case DP_XMM7:   push_xmm(0x54); return;
792                     case DP_MXCSR:  pushfp_mxcsr(); return;
793
794                     default:        val = 0; break;
795             }
796             rpnpush(val);
797             return;
798     }
799
800     /*
801      * Current register main pop function.
802      */
803     static void popr(void)
804     {
805             struct dprobes_struct *dp = &dprobes;
806             unsigned short index = get_u16_oprnd();
807             unsigned long val = rpnpop();
808             switch(index) {
809                     case DP_DS:     dp->regs->xds = val; break;
```

```
810                     case DP_ES:     dp->regs->xes = val; break;
811                     case DP_FS:     write_fs(val); break;
812                     case DP_GS:     write_gs(val); break;
813                     case DP_EAX:    dp->regs->eax = val; break;
814                     case DP_EBX:    dp->regs->ebx = val; break;
815                     case DP_ECX:    dp->regs->ecx = val; break;
816                     case DP_EDX:    dp->regs->edx = val; break;
817                     case DP_EDI:    dp->regs->edi = val; break;
818                     case DP_ESI:    dp->regs->esi = val; break;
819                     case DP_EBP:    dp->regs->ebp = val; break;
820                     default:        break;
821             }
822     }
823
824     /*
825      * User register main pop function.
826      */
827     static void popu(void)
828     {
829             struct dprobes_struct *dp = &dprobes;
830             unsigned short index = get_u16_oprnd();
831             unsigned long val = rpnpop();
832             switch(index) {
833                     case DP_DS:     dp->uregs->xds = val; break;
834                     case DP_ES:     dp->uregs->xes = val; break;
835                     case DP_FS:     write_fs(val); break;
836                     case DP_GS:     write_gs(val); break;
837                     case DP_EAX:    dp->uregs->eax = val; break;
838                     case DP_EBX:    dp->uregs->ebx = val; break;
839                     case DP_ECX:    dp->uregs->ecx = val; break;
840                     case DP_EDX:    dp->uregs->edx = val; break;
841                     case DP_EDI:    dp->uregs->edi = val; break;
842                     case DP_ESI:    dp->uregs->esi = val; break;
843                     case DP_EBP:    dp->uregs->ebp = val; break;
844                     default:        break;
845             }
846     }
847
848     /*
849      * Entry point for the dprobes interpreter(Probe handler).
850      */
851     static void dp_asm_interpreter(byte_t rpn_instr)
852     {
853             struct dprobes_struct *dp = &dprobes;
854
855                     switch(rpn_instr) {
856
857                             case DP_SEG2LIN:        seg2lin(); break;
858
859                             case DP_PUSH_IO_U8:     push_io_u8(); break;
860                             case DP_PUSH_IO_U16:    push_io_u16(); break;
861                             case DP_PUSH_IO_U32:    push_io_u32(); break;
862                             case DP_POP_IO_U8:      pop_io_u8(); break;
863                             case DP_POP_IO_U16:     pop_io_u16(); break;
864                             case DP_POP_IO_U32:     pop_io_u32(); break;
865
866                             default: gen_ex(EX_INVALID_OPCODE, *(dp->rpn_ip - 1),
867                             (unsigned long)(dp->rpn_ip -dp->rpn_code - 1)); break;          }
868             return;
869     }
```

```
#ifndef __ASM_I386_DPROBES_EXCLUDE_H__
#define __ASM_I386_DPROBES_EXCLUDE_H__

/*
 * IBM Dynamic Probes
 * Copyright (c) International Business Machines Corp., 2000
 *
 * 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.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111−1307, USA.
 */

/*
 * Exclude regions markers: We do not allow probes to be placed in these
 * code areas to _prevent_ recursion into dp_trap3. Note that this does
 * not eliminate the possibility of recursion completely. dp_trap3 handles
 * recursion by silently and permanently removing the recursed probe point.
 */
extern void dprobes_code_start(void);
extern void dprobes_code_end(void);
extern void dprobes_asm_code_start(void);
extern void dprobes_asm_code_end(void);
extern void dprobes_hooks_code_start(void);
extern void dprobes_hooks_code_end(void);
extern void dprobes_hooks_asm_code_start(void);
extern void dprobes_hooks_asm_code_end(void);
extern void dprobes_interpreter_code_start(void);
extern void dprobes_interpreter_code_end(void);
extern void dprobes_time_start(void);
extern void dprobes_time_end(void);
extern void dprobes_asm_time_start(void);
extern void dprobes_asm_time_end(void);
#ifdef CONFIG_DR_ALLOC
extern void dr_alloc_asm_start(void);
extern void dr_alloc_asm_end(void);
#endif

struct region {
        unsigned long start;
        unsigned long end;
};

#define NR_EXCLUDED_REGIONS 8
static struct region exclude[] = {
#ifdef CONFIG_DR_ALLOC
        {(unsigned long)dr_alloc_asm_start,
                (unsigned long)dr_alloc_asm_end},
#else
        {0, 0},
#endif
        {(unsigned long)dprobes_code_start,
                (unsigned long)dprobes_code_end},
        {(unsigned long)dprobes_asm_code_start,
                (unsigned long)dprobes_asm_code_end},
        {(unsigned long)dprobes_hooks_code_start,
                (unsigned long)dprobes_hooks_code_end},
        {(unsigned long)dprobes_hooks_asm_code_start,
                (unsigned long)dprobes_hooks_asm_code_end},
        {(unsigned long)dprobes_interpreter_code_start,
                (unsigned long)dprobes_interpreter_code_end},
        {(unsigned long)dprobes_time_start,
                (unsigned long)dprobes_time_end},
        {(unsigned long)dprobes_asm_time_start,
                (unsigned long)dprobes_asm_time_end}
};

static inline int excluded(unsigned long addr)
{
        int i;
        for (i = 0; i < NR_EXCLUDED_REGIONS; i++) {
                if (addr >= exclude[i].start && addr <= exclude[i].end) {
                        return 1;
                }
        }
        return 0;
}

#endif /* __ASM_I386_DPROBES_EXCLUDE_H__ */
```

```
#ifndef __ASM_I386_DPROBES_H__
#define __ASM_I386_DPROBES_H__

/*
 * IBM Dynamic Probes
 * Copyright (c) International Business Machines Corp., 2000
 *
 * 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.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111−1307, USA.
 */

/*
 * RPN stack width will always be equal to the machine register width.
 */
#define RPN_STACK_SIZE          1024
#define CALL_FRAME_SIZE         10
#define NR_NESTED_CALLS         32
/* 1st is a dummy frame, not used by call instruction */
#define CALL_STACK_SIZE         (NR_NESTED_CALLS+1)*CALL_FRAME_SIZE
#define FMT_LOG_HDR_SIZE        256

/* Offsets of different items on call stack */
#define OFFSET_SBP              0
#define OFFSET_GV_RANGE         1
#define OFFSET_LV_RANGE         3
#define OFFSET_RPN_STACK_RANGE  5
#define OFFSET_EX_HANDLER       7
#define OFFSET_CALLED_ADDR      8
#define OFFSET_RETURN_ADDR      9

/*
 * offsets of different repitition entries (3 byte prefix) in the
 * exception stacktrace buffer.
 */
struct ex_buffer_offset {
        unsigned long st;       /* ex stack traces */
        unsigned long sf;       /* ex stack frames */
        unsigned long rpn;      /* rpn stack entries */
        unsigned long lv;       /* lv entries */
        unsigned long gv;       /* gv entries */
};

/* Token bytes of all available 3 byte prefixes */
#define TOKEN_SEG_FAULT         −2
#define TOKEN_MEMORY_FAULT      −1
#define TOKEN_MEMORY_LOG        0
#define TOKEN_ASCII_LOG         1
#define TOKEN_STACK_TRACE       2
#define TOKEN_STACK_FRAME       3
#define TOKEN_RPN_ENTRY         4
#define TOKEN_LV_ENTRY          5
#define TOKEN_GV_ENTRY          6
#define TOKEN_LOG               7

#define PREFIX_SIZE             3

struct dprobes_struct {
        unsigned long status;   /* status */
        /*
         * details about the probe that is hit, kept here for quick access at
         * trap1 time.
         */
        unsigned long probe_addr;
        struct pt_regs *regs;
        struct pt_regs *uregs;
        struct dp_module_struct *mod;
        struct dp_record_struct *rec;

        /*
         * aliasing stuff
         */
        pte_t *alias_pte;
        unsigned long alias_pte_val;
        void *alias_addr;
        unsigned long reset_addr;

        /*
         * eflags save / restore area.
         *
```

```
91              * Both these are initalized with the eflags from the stack frame in
92              * trap 3 handler. saved_eflags is edited to mask out IF, if the
93              * instruction being ss'ed can modify IF.
94              *
95              * saved_eflags is used to restore eflags when the ss'ed instruction
96              * is executed successfully.
97              * old_eflags is used when the single-stepped instruction faults.
98              */
99             unsigned long saved_eflags;
100            unsigned long old_eflags;
101
102            /* fpu registers are saved here */
103            union i387_union fpu_save_area;
104
105            /*
106             * Optional data to store with the log record. This data is collected
107             * in the interpreter and used in the trap1 handler when writing the
108             * log to the specified log target.
109             */
110            unsigned long eip;
111            unsigned short cs;
112            unsigned long esp;
113            unsigned short ss;
114            unsigned short major, minor;
115
116            /*
117             * per-processor data used by the interpreter.
118             */
119            unsigned long rpn_tos, call_tos, log_len, prev_log_len, ex_log_len;
120            byte_t * rpn_code;
121            byte_t * rpn_ip;
122            unsigned short jmp_count;
123            byte_t opcode;
124            byte_t reserved;
125            byte_t ex_pending;
126            unsigned long ex_code;
127            unsigned long ex_parm1, ex_parm2, ex_hand;
128            struct ex_buffer_offset ex_off;
129            unsigned long rpn_sbp;
130
131            /*
132             * locks used to handle recursion.
133             */
134            spinlock_t lock;
135            long sem;
136
137            unsigned long rpn_stack[RPN_STACK_SIZE];
138            unsigned long call_stack[CALL_STACK_SIZE];
139            unsigned char log[LOG_SIZE];
140            unsigned char ex_log[EX_LOG_SIZE];
141            unsigned char log_hdr[FMT_LOG_HDR_SIZE];
142    };
143
144    /*
145     * status
146     *
147     * DP_KENREL_PROBE: When this is set, we will not use the pte* fields in the
148     * alias structure at trap1 time, we can write directly to the probe_addr.
149     *
150     * DP_STATUS_ERROR: Contains all the bits to check for error conditions.
151     *
152     * DP_STATUS_DONE: Contains all the per-probe hit bits that need to be turned
153     * off after a probe handler is executed.
154     *
155     * DP_STATUS_ABORT: Indicates that the designated exit facility should not
156     * be called after the original instruction is successfully single-stepped.
157     */
158    #define DP_STATUS_INACTIVE      0x00000000
159    #define DP_STATUS_ACTIVE        0x00000001
160
161    #define DP_STATUS_ERROR         0x00ff0000
162    #define DP_STATUS_GPF           0x00010000
163    #define DP_STATUS_PF            0x00020000
164    #define DP_STATUS_LOG_OVERFLOW  0x00040000
165
166    #define DP_STATUS_INTERPRETER   0x01000000
167    #define DP_STATUS_SS            0x02000000
168    #define DP_KERNEL_PROBE         0x04000000
169    #define DP_USER_PROBE           0x08000000
170    #define DP_STATUS_ABORT         0x10000000
171    #define DP_STATUS_FIRSTFPU      0x20000000
172    #define DP_STATUS_DONE          0xffff0000
173
174    #define DP_INSTR_BREAKPOINT     0xcc
175
176    /* arch-specific rec flags */
177    #define DP_REC_OPCODE_EMULATED          0x01000000
178    #define DP_REC_OPCODE_IF_MODIFIER       0x02000000
179
180    #ifndef EF_CF
```

```
#define EF_CF   0x00000001
#endif
#ifndef EF_TF
#define EF_TF   0x00000100
#endif
#ifndef EF_IE
#define EF_IE   0x00000200
#endif
#ifndef EF_DF
#define EF_DF   0x00000400
#endif
#ifndef EF_RF
#define EF_RF   0x00010000
#endif

#define CR0_TS  0x00000008

#ifndef HAVE_HWFP
#ifdef CONFIG_MATH_EMULATION
#define HAVE_HWFP (boot_cpu_data.hard_math)
#else
#define HAVE_HWFP 1
#endif
#endif

/*
 *  Function declarations
 */
extern void dp_interpreter(void);
extern int dp_gpf(struct pt_regs *);
extern int dp_pf(struct pt_regs *);
extern int dp_handle_fault(struct pt_regs *);
extern int dp_do_debug(struct pt_regs *, unsigned long);
extern int dp_trap1(struct pt_regs *);
extern int dp_trap(struct pt_regs *, int, unsigned int, unsigned long);
extern int dp_seg_to_flat(unsigned long, unsigned long, void **);
extern int __remove_probe(byte_t *, struct dp_record_struct *);
extern int __insert_probe(byte_t *, struct dp_record_struct *, struct dp_module_struct *, struct page *);

#endif
```

```
#ifndef __ASM_I386_DPROBES_IN_H__
#define __ASM_I386_DPROBES_IN_H__

/*
 * IBM Dynamic Probes
 * Copyright (c) International Business Machines Corp., 2000
 *
 * 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.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111−1307, USA.
 */

/*
 * This header file defines the opcodes for the RPN instructions
 *
 * Opcodes 0x00 to 0xAF are for common, arch−independent instructions.
 * Opcodes 0xB0 to 0xDF are for arch−dependent instructions.
 * Opcodes 0xF1 to 0xFF are for two byte instructions.
 */

/*
 * Misc
 */
#define DP_SEG2LIN              0xB0

/*
 * IO Group.
 */
#define DP_PUSH_IO_U8           0xB1
#define DP_PUSH_IO_U16          0xB2
#define DP_PUSH_IO_U32          0xB3
#define DP_POP_IO_U8            0xB4
#define DP_POP_IO_U16           0xB5
#define DP_POP_IO_U32           0xB6

/*
 * Intel32 Register Assignments.
 */
#define DP_CS                   0x0000
#define DP_DS                   0x0001
#define DP_ES                   0x0002
#define DP_FS                   0x0003
#define DP_GS                   0x0004
#define DP_SS                   0x0005
#define DP_EAX                  0x0006
#define DP_EBX                  0x0007
#define DP_ECX                  0x0008
#define DP_EDX                  0x0009
#define DP_EDI                  0x000a
#define DP_ESI                  0x000b
#define DP_EFLAGS               0x000c
#define DP_EIP                  0x000d
#define DP_ESP                  0x000e
#define DP_EBP                  0x000f

#define DP_TR                   0x0020
#define DP_LDTR                 0x0021
#define DP_GDTR                 0x0022
#define DP_IDTR                 0x0023
#define DP_CR0                  0x0024
#define DP_CR1                  0x0025
#define DP_CR2                  0x0026
#define DP_CR3                  0x0027
#define DP_CR4                  0x0028

#define DP_DR0                  0x002c
#define DP_DR1                  0x002d
#define DP_DR2                  0x002e
#define DP_DR3                  0x002f
#define DP_DR4                  0x0030
#define DP_DR5                  0x0031
#define DP_DR6                  0x0032
#define DP_DR7                  0x0033

#define DP_CPUID                0x003c
#define DP_MSR                  0x003d

#define DP_FR0                  0x003e
#define DP_FR1                  0x003f
#define DP_FR2                  0x0040
```

```
91  #define DP_FR3                  0x0041
92  #define DP_FR4                  0x0042
93  #define DP_FR5                  0x0043
94  #define DP_FR6                  0x0044
95  #define DP_FR7                  0x0045
96  #define DP_FCW                  0x0046
97  #define DP_FSW                  0x0047
98  #define DP_FTW                  0x0048
99  #define DP_FIP                  0x0049
100 #define DP_FCS                  0x004a
101 #define DP_FDP                  0x004b
102 #define DP_FDS                  0x004c
103
104 #define DP_XMM0                 0x004d
105 #define DP_XMM1                 0x004e
106 #define DP_XMM2                 0x004f
107 #define DP_XMM3                 0x0050
108 #define DP_XMM4                 0x0051
109 #define DP_XMM5                 0x0052
110 #define DP_XMM6                 0x0053
111 #define DP_XMM7                 0x0054
112 #define DP_MXCSR                0x0055
113
114 #ifndef __KERNEL__
115 #ifndef PAGE_OFFSET
116 #define PAGE_OFFSET             0xc0000000
117 #endif
118 #endif /* !__KERNEL__ */
119
120 #endif
```

```
#ifndef _LINUX_DPROBES_H
#define _LINUX_DPROBES_H

/*
 * IBM Dynamic Probes
 * Copyright (c) International Business Machines Corp., 2000
 *
 * 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.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111−1307, USA.
 */

#ifdef __KERNEL__
#include <linux/types.h>
#else
#include <sys/types.h>
#endif

#define DP_MAJOR_VER    3
#define DP_MINOR_VER    6
#define DP_PATCH_VER    3

/* main command codes */
#define DP_CMD_MASK             0x000000ff
#define DP_INSERT               0x00000001
#define DP_REMOVE               0x00000002
#define DP_GETVARS              0x00000004
#define DP_QUERY                0x00000008
#define DP_HELP                 0x00000010
#define DP_VERSION              0x00000020
#define DP_BUILDPPDF            0x00000040
#define DP_APPLYPPDF            0x00000080

/* command modifiers */
#define DP_FLAGS_MASK           0xff000000
#define DP_ALL                  0x80000000

/* insert cmd modifiers */
#define DP_MERGE                0x01000000
#define DP_REPLACE              0x02000000
#define DP_STOP_CPUS            0x04000000
#define DP_DONT_VERIFY_OPCODES  0x08000000
#define DP_AUTOSTACKTRACE       0x10000000

/* getvars cmd modifiers */
#define DP_GETVARS_INDEX        0x01000000
#define DP_GETVARS_RESET        0x02000000
#define DP_GETVARS_LOCAL        0x04000000
#define DP_GETVARS_GLOBAL       0x08000000

/* query cmd modifiers */
#define DP_QUERY_EXTENDED       0x01000000

/* pgm−>flags: default data that will be part of each log record */
#define DP_LOG_MASK             0x0000ff00
#define DP_LOG_PROCNAME         0x00000100
#define DP_LOG_PID              0x00000200
#define DP_LOG_UID              0x00000400
#define DP_LOG_NIP             0x00000800
#define DP_LOG_PSW             0x00000800
#define DP_LOG_UPSW            0x00001000
#define DP_LOG_SS_ESP           0x00000800
#define DP_LOG_CS_EIP           0x00001000
#define DP_LOG_TSC              0x00002000
#define DP_LOG_CPU              0x00004000
#define DP_LOG_ALL              0x0000ff00

/* pgm−>flags: default log target */
#define DP_LOG_TARGET_MASK      0x00ff0000
#define DP_LOG_TARGET_KLOG      0x00010000
#define DP_LOG_TARGET_LTT       0x00020000
#define DP_LOG_TARGET_COM1      0x00040000
#define DP_LOG_TARGET_COM2      0x00080000
#define DP_LOG_TARGET_EVL       0x00100000

/* exit facilities for use with "exit_to" instruction */
#define DP_EXIT_TO_SGI_KDB      0x01
#define DP_EXIT_TO_SGI_VMDUMP   0x02
#define DP_EXIT_TO_CORE_DUMP    0x03

```

```
/* bit flags to indicate elements present in the trace buffer header. */
#define DP_HDR_MAJOR            0x00000001
#define DP_HDR_MINOR            0x00000002
#define DP_HDR_CPU              0x00000004
#define DP_HDR_PID              0x00000008
#define DP_HDR_UID              0x00000010
#define DP_HDR_CS               0x00000020
#define DP_HDR_EIP              0x00000040
#define DP_HDR_SS               0x00000080
#define DP_HDR_ESP              0x00000100
#define DP_HDR_TSC              0x00000200
#define DP_HDR_PROCNAME         0x00000400

typedef unsigned char byte_t;
/* FIXME: Well, we're working on getting this in arch headers */
#if defined(CONFIG_X86) || defined(CONFIG_IA64)
typedef byte_t opcode_t;
#endif
#ifdef CONFIG_ARCH_S390X
typedef u_int16_t  opcode_t;
#elif CONFIG_ARCH_S390
typedef u_int16_t  opcode_t;
#endif
#ifdef CONFIG_PPC
typedef u_int32_t opcode_t;
#endif

/*
 * This captures the header information of each probe point. It also contains
 * links to the rpn code of this dprobe handler.
 *
 * maxhits: This field indicates the number of times this probe will be
 * executed before being disabled. A negative value here means that the
 * probe will not be disabled automatically.
 *
 * count: This field keeps track of number of times this probe is hit,
 * and executed successfully. There is a reason this is "long" and not
 * "unsigned long". We would implement "pass-count" feature that specifies
 * the number of times we pass over this probe with actually executing the
 * probe handler, using the fact that count is a signed value.
 */
struct dp_point_struct {
        loff_t offset;  /* file offset or absolute address */
        unsigned short address_flag;
        opcode_t opcode;
        opcode_t actual_opcode; /* in case of opcode mismatch */
        unsigned short major;
        unsigned short minor;
        unsigned short group;
        unsigned short type;
        unsigned long rpn_offset; /* offset into rpn_code of dp_pgm_struct */
        unsigned long rpn_end;
        unsigned short probe; /* watchpoint info, probe and access types */
        unsigned long len;      /* range of the watchpoint probe */
        int dbregno;    /* the debug reg used for this watchpoint */
        long maxhits;
        long passcount;
        unsigned char logonfault;
        unsigned short ex_mask;
};

#define DP_ADDRESS_ABSOLUTE     0x0001

/*
 * Flags related to watchpoint probes.
 */
#define DP_PROBE_BREAKPOINT     0x8000
#define DP_PROBE_WATCHPOINT     0x4000
#define DP_PROBE_MASK           0xc000

#define DP_WATCHTYPE_EXECUTE    0x0000
#define DP_WATCHTYPE_WRITE      0x0001
#define DP_WATCHTYPE_IO         0x0002
#define DP_WATCHTYPE_RDWR       0x0003
#define DP_WATCHTYPE_MASK       0x0003

/*
 * This captures the information specified in the dprobe program file header.
 *
 * User application passes the details of the pgm header to the system call in
 * this structure. It is also used in the kernel as part of dp_module_struct.
 */
struct dp_pgm_struct {
        unsigned char *name; /* module for which the probe program is written */
        unsigned long flags;
        unsigned short major;
        unsigned long id;
        unsigned short jmpmax;
        unsigned short logmax;
        unsigned short ex_logmax; /* length of exception stack trace buffer */
```

```
181         unsigned short num_lv;  /* num of local variables */
182         unsigned short num_gv;  /* num of global variables */
183
184         unsigned short rpn_length; /* rpn code */
185         unsigned char autostacktrace;
186         byte_t *rpn_code;
187
188         unsigned short align; /* kernel module alignment */
189         unsigned short num_points;
190         struct dp_point_struct *point;
191 };
192
193 /* pgm flags */
194 #define DP_MODTYPE_MASK         0x000f
195 #define DP_MODTYPE_USER         0x0001
196 #define DP_MODTYPE_KERNEL       0x0002
197 #define DP_MODTYPE_KMOD         0x0004
198
199 #define LOG_SIZE                1024
200 #define EX_LOG_SIZE             1024
201 #define JMP_MAX                 65535
202 #define MAX_MAXHITS             0x7fffffff
203
204 #define DEFAULT_LOGMAX          LOG_SIZE
205 #define DEFAULT_EX_LOG_MAX      EX_LOG_SIZE
206 #define DEFAULT_JMPMAX          JMP_MAX
207 #define DEFAULT_MAXHITS         MAX_MAXHITS
208
209 /* compiled in limits */
210 #define DP_MAX_LVARS            256
211 #define DP_MAX_GVARS            256
212
213 /* exception codes */
214 #define EX_INVALID_ADDR         0x0001
215 #define EX_SEG_FAULT            0x0002
216 #define EX_MAX_JMPS             0x0004
217 #define EX_CALL_STACK_OVERFLOW  0x0010
218 #define EX_DIV_BY_ZERO          0x0020
219 #define EX_INVALID_OPERAND      0x0040
220 #define EX_INVALID_OPCODE       0x0080
221 #define EX_LOG_OVERFLOW         0x1000
222 #define EX_RPN_STACK_WRAP       0x2000
223 #define EX_USER                 0x8000
224 #define DEFAULT_EX_MASK         0x0fff
225
226 /* If a non−maskable ex is masked out, the interpreter will be terminated. */
227 #define EX_NON_MASKABLE_EX      0x0fff
228
229 #define EX_NO_HANDLER           0xffffffff
230
231 /*
232  * This defines the Dynamic Probe getvars Request Packet, passed in from
233  * the user for subfunction DP_GETVARS.
234  */
235 struct  dp_getvars_in_struct  {
236         unsigned char *name;    /* loadable module name */
237         unsigned short from;
238         unsigned short to;
239         unsigned long allocated; /* Allocated size of result pkt */
240         unsigned long returned; /* Returned size of result pkt */
241 };
242
243 /*
244  * The output from getvars command is returned in this format. First all
245  * global variables will be there as dp_getvars_global_out_struct followed
246  * by local variables for each module in form of dp_getvars_local_out_struct.
247  */
248 struct  dp_getvars_global_out_struct  {
249         unsigned long num_vars; /* number of the variables to follow */
250         /* variables */
251 };
252
253 struct dp_getvars_local_out_struct {
254         unsigned long length;  /* length of this element */
255         unsigned long num_vars; /* number of the variables to follow */
256         /* variables followed by the name of loadable module*/
257 };
258
259 /*
260  * This defines the Dynamic Probe query Request Packet passed in
261  * from the user for subfunction DP_QUERY.
262  */
263 struct dp_query_in_struct {
264         unsigned char *name; /* loadable module name */
265         unsigned long allocated; /* Allocated size of result pkt */
266         unsigned long returned; /* Returned size of result pkt */
267 };
268
269 struct dp_outrec_struct {
270         unsigned long status;
```

```
271         long count;
272         int dbregno;    /* the debug reg used for this watchpoint */
273         struct dp_point_struct point;
274 };
275
276 /*
277  * flags
278  *
279  * Initially a probe record starts out with zero flags.
280  *
281  * DP_REC_STATUS_COMPILED: A probe record is compiled, after successfully
282  * validating the rpn code.
283  *
284  * DP_REC_STATUS_ACTIVE: A probe record becomes valid after it is verified
285  * that the opcode as specified in the probe program matches with the one
286  * at the specified location. We would probably do this verification the
287  * first time this probe is inserted.
288  *
289  * DP_REC_STATUS_MISMATCH: This means the opcode specified by the probe
290  * does not match with the one present when we try to insert it.
291  *
292  * DP_REC_STATUS_DISABLED: A probe will be disabled after being executed
293  * for max-hit number of times automatically.
294  *
295  * DP_REC_STATUS_REMOVED: The status of a probe will be removed if user
296  * explicitly removes it using dprobe --remove --major-minor command.
297  * Note that if a specific probe is removed by its major-minor, we
298  * don't remove the dp_record_struct from memory as it is too much
299  * work. We simply note that fact in the status flags and leave the
300  * dp_record_struct alone. But, if an entire probe program is removed,
301  * all its structures will be removed permanently.
302  *
303  * DP_REC_STATUS_EXCLUDED: This means that the probe falls within the
304  * DProbes defined exclude regions.
305  *
306  * DP_REC_STATUS_EXCLUDED_OPCODE: Set for probes on opcodes on which probes are
307  * are not allowed.
308  *
309  * DP_REC_STATUS_INVALID_OFFSET: Set if the address specified for probe insertion
310  * is out of bounds for that executable module.
311  *
312  * DP_REC_STATUS_DEBUGREG_UNAVAIL: Set if the probe is of type watchpoint and
313  * no free debug registers are free for use.
314  *
315  * DP_REC_STATUS_WATCHPOINT_LEN_INVALID: Set if the probe is of type watchpoint
316  * and the range of address specified is not valid.
317  */
318 #define DP_REC_STATUS_COMPILED                  0x00000001
319 #define DP_REC_STATUS_ACTIVE                    0x00000002
320 #define DP_REC_STATUS_MISMATCH                  0x00000004
321 #define DP_REC_STATUS_DISABLED                  0x00000008
322 #define DP_REC_STATUS_REMOVED                   0x00000010
323 #define DP_REC_STATUS_EXCLUDED                  0x00000020
324 #define DP_REC_STATUS_EXCLUDED_OPCODE           0x00000040
325 #define DP_REC_STATUS_INVALID_OFFSET            0x00000080
326 #define DP_REC_STATUS_DEBUGREG_UNAVAIL          0x00000100
327 #define DP_REC_STATUS_WATCHPOINT_LEN_INVALID    0x00000200
328 #define DP_REC_STATUS_DEBUGREG_PATCH_NEEDED     0x00000400
329 #define DP_REC_ARCH_FLAGS                       0xff000000
330
331 struct dp_outmod_struct {
332         struct dp_outmod_struct *next;  /* link */
333         unsigned long flags;    /* cmdline switches */
334         struct dp_outrec_struct * rec;  /* array of dp_outrec_structs. */
335         unsigned long * lv;     /* local variables */
336         struct dp_pgm_struct pgm;
337         unsigned long base;
338         unsigned long end;
339 };
340
341 struct dp_ioctl_arg {
342         void * input;
343         void * output;
344         unsigned long cmd;
345 };
346
347 #ifdef __KERNEL__
348
349 #include <linux/sched.h>
350 #include <linux/mm.h>
351 #include <asm/page.h>
352 #include <asm/pgtable.h>
353 #include <asm/ptrace.h>
354
355 /*
356  * In addition to the header information of each probe point, we also have hash
357  * chains to get to the dprobe record quickly in the kernel at trap3 time.
358  */
359 struct dp_record_struct {
360         struct dp_record_struct *next_hash;
```

```
361         struct dp_record_struct **pprev_hash;
362         struct dp_module_struct *mod;
363         spinlock_t lock;
364         unsigned long status;
365         long count;
366         int dbregno;    /* the debug reg used for this watchpoint */
367         struct dp_point_struct point;
368 };
369
370 /*
371  * When Dprobes stores it's log in a trace buffer, the log data is preceded
372  * by the header information as in this structure, followed by optional
373  * header elements.
374  */
375 #define DP_TRACE_HDR_ID 1
376 struct dp_trace_hdr_struct {
377         unsigned short facility_id;
378         unsigned short len;
379         unsigned long mask;
380         unsigned short major;
381         unsigned short minor;
382 };
383
384 #ifdef CONFIG_SMP
385 #define MIN_ST_SIZE     sizeof(struct dp_trace_hdr_struct) + sizeof(int)
386 #else
387 #define MIN_ST_SIZE     sizeof(struct dp_trace_hdr_struct)
388 #endif
389
390 /* Major number 0 is reserved for stack trace log record */
391 #define DP_ST_MAJOR     0x0000
392 #define DP_ST_MINOR     0x0000
393
394 /*
395  * This is the data structure that we maintain in kernel for each module
396  * that has any probes applied on it.
397  */
398 struct dp_module_struct {
399         struct dp_module_struct *next;  /* link */
400         unsigned long flags;    /* cmdline switches */
401         struct dp_record_struct * rec;  /* array of dp_record_structs. */
402         unsigned long * lv;     /* local variables */
403         struct inode * inode;   /* for quicker access to inode */
404         int trace_id;           /* ltt trace event id */
405         struct dp_trace_hdr_struct hdr;
406         struct dp_pgm_struct pgm;
407         unsigned long base; /* used to store kmod base address */
408         unsigned long end; /* used to store kmod base address */
409         struct nameidata nd; /* to hold path/dentry etc. */
410         struct address_space_operations * ori_a_ops;
411         struct address_space_operations dp_a_ops;
412 };
413
414 #ifdef CONFIG_SMP
415 extern struct dprobes_struct dprobes_set[];
416 #define dprobes dprobes_set[smp_processor_id()]
417 #else
418 extern struct dprobes_struct dprobes;
419 #define dprobes_set (&dprobes)
420 #endif /* CONFIG_SMP */
421
422 extern char _stext, _etext; /* kernel start and end */
423
424 /*
425  * global variables stuff
426  */
427 extern unsigned long *dp_gv;
428 extern unsigned long dp_num_gv;
429 extern rwlock_t dp_gv_lock;
430
431 /*
432  * dp_record_struct hashing, based on page cache hashing.
433  */
434 #define REC_HASH_BITS 12
435 #define REC_HASH_SIZE (1 << REC_HASH_BITS)
436
437 extern struct dp_record_struct *dp_rec_hash_table[REC_HASH_SIZE];
438 extern rwlock_t dp_rechash_lock;
439
440 extern inline unsigned long _rec_hashfn(struct inode * inode, unsigned long offset)
441 {
442 #define i (((unsigned long) inode)/(sizeof(struct inode) & ~ (sizeof(struct inode) - 1)))
443 #define s(x) ((x)+((x)>>REC_HASH_BITS))
444         return s(i+offset) & (REC_HASH_SIZE-1);
445 #undef i
446 #undef o
447 #undef s
448 }
449
450 #define rec_hash(inode,offset) (dp_rec_hash_table + _rec_hashfn(inode,offset))
```

```
451
452 /* It returns with rec->lock held. */
453 static inline struct dp_record_struct * __find_rec(struct inode * inode, unsigned long offset, struct dp_record_s
    truct *rec)
454 {
455         unsigned long eflags;
456
457         read_lock_irqsave(&dp_rechash_lock, eflags);
458         goto inside;
459         for (;;) {
460                 rec = rec->next_hash;
461 inside:
462                 if (!rec)
463                         goto not_found;
464                 if (inode) {
465                         if (rec->mod->inode != inode)
466                                 continue;
467                 }
468                 else {
469                         if (rec->mod->inode)
470                                 continue;
471                 }
472                 if (rec->point.offset == offset) {
473                         spin_lock(&rec->lock);
474                         break;
475                 }
476         }
477         /* Found the record */
478 not_found:
479         read_unlock_irqrestore(&dp_rechash_lock, eflags);
480         return rec;
481 }
482
483 static inline struct dp_record_struct *find_rec(struct inode * inode, unsigned long offset)
484 {
485         return __find_rec(inode, offset, *rec_hash(inode, offset));
486 }
487
488 static inline void remove_rec_from_hash_queue(struct dp_record_struct * rec)
489 {
490         unsigned long eflags;
491
492         write_lock_irqsave(&dp_rechash_lock, eflags);
493         spin_lock(&rec->lock);
494         if(rec->pprev_hash) {
495                 if(rec->next_hash)
496                         rec->next_hash->pprev_hash = rec->pprev_hash;
497                 *rec->pprev_hash = rec->next_hash;
498                 rec->pprev_hash = NULL;
499         }
500         write_unlock_irqrestore(&dp_rechash_lock, eflags);
501         spin_unlock(&rec->lock);
502 }
503
504 static inline void __add_rec_to_hash_queue(struct dp_record_struct * rec, struct dp_record_struct **p)
505 {
506         unsigned long eflags;
507
508         write_lock_irqsave(&dp_rechash_lock, eflags);
509         spin_lock(&rec->lock);
510         if (rec->pprev_hash) /* already hashed */
511                 goto done;
512         if((rec->next_hash = *p) != NULL)
513                 (*p)->pprev_hash = &rec->next_hash;
514         *p = rec;
515         rec->pprev_hash = p;
516 done:
517         write_unlock_irqrestore(&dp_rechash_lock, eflags);
518         spin_unlock(&rec->lock);
519 }
520
521 static inline void add_rec_to_hash_queue(struct dp_record_struct * rec, struct inode * inode, unsigned long offse
    t)
522 {
523         __add_rec_to_hash_queue(rec, rec_hash(inode,offset));
524 }
525
526 extern int dp_insmod(struct module *kmod);
527 extern int dp_remmod(struct module *kmod);
528 extern int dp_readpage(struct file *, struct page *);
529 extern int dp_writepage(struct page *page);
530
531 #define IS_COW_PAGE(page, inode) (!(page->mapping) || \
532         (page->mapping->host != (void *)inode))
533
534 #include <asm/dprobes.h>
535 #include <linux/dprobes_hooks.h>
536
537 #endif /* __KERNEL__ */
538
```

```
539 #endif
```

```
#ifndef _LINUX_DPROBES_IN_H
#define _LINUX_DPROBES_IN_H

/*
 * IBM Dynamic Probes
 * Copyright (c) International Business Machines Corp., 2000
 *
 * 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.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111−1307, USA.
 */

/*
 * This header file defines the opcodes for the RPN instructions.
 *
 * Opcodes 0x00 to 0xAF are for common, arch−independent instructions.
 * Opcodes 0xB0 to 0xDF are for arch−dependent instructions.
 * Opcodes 0xF1 to 0xFF are for two byte instructions.
 */

#define DP_NOP                  0x00

/*
 * RPN execution group.
 */
#define DP_JMP                  0x01
#define DP_JLT                  0x02
#define DP_JLE                  0x03
#define DP_JGT                  0x04
#define DP_JGE                  0x05
#define DP_JZ                   0x06
#define DP_JNZ                  0x07
#define DP_LOOP                 0x08
#define DP_CALL                 0x09
#define DP_RET                  0x0a
#define DP_ABORT                0x0b
#define DP_REM                  0x0c
#define DP_EXIT                 0x0d
#define DP_EXIT_N               0x0e
#define DP_SUSPEND              0x0f

/*
 * Logging Group.
 */
#define DP_RESUME               0x10
#define DP_SETMIN_I             0x11
#define DP_SETMIN               0x12
#define DP_SETMAJ_I             0x13
#define DP_SETMAJ               0x14
#define DP_LOG_STR              0x15
#define DP_LOG_MRF              0x17
#define DP_LOG_I                0x1b

/*
 * Global Variable Group.
 */
#define DP_ALLOC_GV             0x1c
#define DP_FREE_GV              0x1d
#define DP_FREE_GVI             0x1e
#define DP_PUSH_GVI             0x1f
#define DP_PUSH_GV              0x20
#define DP_POP_GVI              0x21
#define DP_POP_GV               0x22
#define DP_MOVE_GVI             0x23
#define DP_MOVE_GV              0x24
#define DP_INC_GVI              0x25
#define DP_INC_GV               0x26
#define DP_DEC_GVI              0x27
#define DP_DEC_GV               0x28

/*
 * Local Variable Group.
 */
#define DP_PUSH_LVI             0x29
#define DP_PUSH_LV              0x2a
#define DP_POP_LVI              0x2b
#define DP_POP_LV               0x2c
#define DP_MOVE_LVI             0x2d
#define DP_MOVE_LV              0x2e
#define DP_INC_LVI              0x2f
```

```
91  #define DP_INC_LV               0x30
92  #define DP_DEC_LVI              0x31
93  #define DP_DEC_LV               0x32
94
95  /*
96   * Arithmetic Group.
97   */
98  #define DP_ADD                  0x33
99  #define DP_SUB                  0x34
100 #define DP_MUL                  0x35
101
102 /*
103  * Logic Group.
104  */
105 #define DP_NEG                  0x36
106 #define DP_AND                  0x37
107 #define DP_OR                   0x38
108 #define DP_XOR                  0x39
109 #define DP_ROL_I                0x3a
110 #define DP_ROL                  0x3b
111 #define DP_ROR_I                0x3c
112 #define DP_ROR                  0x3d
113 #define DP_SHL_I                0x3e
114 #define DP_SHL                  0x3f
115 #define DP_SHR_I                0x40
116 #define DP_SHR                  0x41
117
118 /*
119  * RPN Stack Group.
120  */
121 #define DP_XCHG                 0x42
122 #define DP_DUP_I                0x43
123 #define DP_DUP                  0x44
124 #define DP_ROS                  0x45
125
126 /*
127  * Register Group.
128  */
129 #define DP_PUSH_R               0x46
130 #define DP_POP_R                0x47
131 #define DP_PUSH_U               0x48
132 #define DP_POP_U                0x49
133
134 /*
135  * Data Group.
136  */
137 #define DP_PUSH                 0x4c
138 #define DP_PUSH_MEM_U8          0x4d
139 #define DP_PUSH_MEM_U16         0x4e
140 #define DP_PUSH_MEM_U32         0x4f
141 #define DP_PUSH_MEM_U64         0x50
142 #define DP_POP_MEM_U8           0x51
143 #define DP_POP_MEM_U16          0x52
144 #define DP_POP_MEM_U32          0x53
145 #define DP_POP_MEM_U64          0x54
146
147 /*
148  * System Variable Group.
149  */
150 #define DP_PUSH_TASK            0x5d
151 #define DP_PUSH_PID             0x5e
152 #define DP_PUSH_PROCID          0x5f
153
154 /*
155  * Address Verification.
156  */
157 #define DP_VFY_R                0x60
158 #define DP_VFY_RW               0x61
159
160 /*
161  * Some more arithmetic/logic instructions.
162  */
163 #define DP_DIV                  0x62
164 #define DP_IDIV                 0x63
165 #define DP_PBL                  0x64
166 #define DP_PBR                  0x65
167 #define DP_PBL_I                0x66
168 #define DP_PBR_I                0x67
169 #define DP_PZL                  0x68
170 #define DP_PZR                  0x69
171 #define DP_PZL_I                0x6a
172 #define DP_PZR_I                0x6b
173
174 /*
175  * Exception handling/Stacktrace instructions.
176  */
177 #define DP_SX                   0x6c
178 #define DP_UX                   0x6d
179 #define DP_RX                   0x6e
180 #define DP_PUSH_X               0x6f
```

```
181 #define DP_PUSH_LP              0x70
182 #define DP_PUSH_PLP             0x71
183 #define DP_POP_LP               0x72
184 #define DP_LOG_ST               0x73
185 #define DP_PURGE_ST             0x74
186 #define DP_TRACE_LV             0x75
187 #define DP_TRACE_GV             0x76
188 #define DP_TRACE_PV             0x77
189
190 #define DP_PUSH_SBP             0x78
191 #define DP_POP_SBP              0x79
192 #define DP_PUSH_TSP             0x7a
193 #define DP_POP_TSP              0x7b
194 #define DP_PUSH_SBP_I           0x7c
195 #define DP_POP_SBP_I            0x7d
196 #define DP_PUSH_TSP_I           0x7e
197 #define DP_POP_TSP_I            0x7f
198 #define DP_COPY_SBP_I           0x80
199 #define DP_COPY_TSP_I           0x81
200 #define DP_PUSH_STP             0x82
201 #define DP_POP_STP              0x83
202
203 #define DP_SAVE_SBP             0x84
204 #define DP_RESTORE_SBP          0x85
205 #define DP_SAVE_TSP             0x86
206 #define DP_RESTORE_TSP          0x87
207 #define DP_COPY_SBP             0x88
208 #define DP_COPY_TSP             0x89
209
210 /* stack based log instructions */
211 #define DP_LOG                  0x8a
212 #define DP_LOG_LV               0x8b
213 #define DP_LOG_GV               0x8c
214
215 #define DP_CALLK                0x8d
216
217 #include <asm/dprobes_in.h>
218
219 #endif
```

Table of Content

End of Table of Contents