90

```
/*
 * Linux Event Logging for the Enterprise
 * Copyright (c) International Business Machines Corp., 2001
 *
 *
 * 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.
 *
 *  Please send e−mail to lkessler@users.sourceforge.net if you have
 *  questions or comments.
 *
 *  Project Website:  http://evlog.sourceforge.net/
 *
 */

#ifndef _LINUX_EVL_LOG_H
#define _LINUX_EVL_LOG_H

#ifndef __KERNEL__
#ifdef _POSIX_THREADS
#include <pthread.h>
#endif
#endif

/* Values for log_flags member */
#define POSIX_LOG_TRUNCATE  0x1
#define EVL_KERNEL_EVENT        0x2
#define EVL_INITIAL_BOOT_EVENT  0x4
#define EVL_KERNTIME_LOCAL      0x8
#define EVL_INTERRUPT           0x10    /* Logged from interrupt context */
#define EVL_PRINTK              0x20    /* Logged by printk() */

/* Formats for optional portion of record. */
#define POSIX_LOG_NODATA    0
#define POSIX_LOG_BINARY    1
#define POSIX_LOG_STRING    2

/* Maximum length of variable portion of record */
#define POSIX_LOG_ENTRY_MAXLEN      (8 * 1024)

/* Maximum length for a string returned by posix_log_memtostr */
/* Thus also the max length of a facility name */
#define POSIX_LOG_MEMSTR_MAXLEN 128

typedef unsigned int posix_log_facility_t;
typedef int posix_log_severity_t;
typedef int posix_log_recid_t;
typedef int posix_log_procid_t;

#define EVL_INVALID_FACILITY ((posix_log_facility_t)−1)

struct posix_log_entry {
        unsigned int            log_magic;
        posix_log_recid_t       log_recid;
        size_t                  log_size;
        int                     log_format;
        int                     log_event_type;
        posix_log_facility_t    log_facility;
        posix_log_severity_t    log_severity;
        uid_t                   log_uid;
        gid_t                   log_gid;
        pid_t                   log_pid;
        pid_t                   log_pgrp;
        struct timespec         log_time;
        unsigned int            log_flags;
#ifdef __KERNEL__
        unsigned long int       log_thread;
#else
#ifdef _POSIX_THREADS
        pthread_t               log_thread;
#else
        unsigned long int       log_thread;
#endif
#endif
        posix_log_procid_t      log_processor;
};

typedef struct posix_log_entry rec_hdr_t;
typedef struct evl_buf_rec {
```

```
91          struct posix_log_entry  rechdr;
92          char                    varbuf[1];
93  } evl_buf_rec_t;
94
95
96  #define LOGFILE_MAGIC   0xbeefface
97  #define LOGREC_MAGIC    0xfeefface
98  #define REC_HDR_SIZE    sizeof(struct posix_log_entry)
99
100 /*
101  * Reserved Event Types
102  */
103 #define EVL_SYSLOG_MESSAGE      0x1
104 #define EVL_PRINTK_MESSAGE      0x2
105 #define EVL_BUFFER_OVERRUN      0x6
106 #define EVL_DUPS_DISCARDED      0x7
107
108 /*
109  * A record of type (LOG_LOGMGMT, EVLOG_REGISTER_FAC) is generated by
110  * evl_register_facility().
111  */
112 #define EVLOG_REGISTER_FAC 40
113
114 #define LOG_LOGMGMT             (12<<3)        /* EVL Facility */
115
116 /*
117  * The optional portion of a record of type (LOG_LOGMGMT, EVLOG_REGISTER_FAC)
118  * is an evl_facreg_rq object followed by a string (the facility name).
119  * evlogd intercepts this record and does the requested registration, if needed.
120  *
121  * fr_kernel_fac_code is the facility code generated by the kernel's call
122  * to evl_register_facility().  This field is filled in by the kernel.
123  *
124  * fr_registry_fac_code is the facility code that appears in the registry.
125  * (This should match fr_kernel_fac_code.)  This field is filled in by
126  * evlogd after it intercepts and executes this request.
127  *
128  * fr_rq_status is the request's status:
129  *      frst_kernel_failed: Set by the kernel to indicate that it could not
130  *              generate a valid facility code for the given name.
131  *      frst_kernel_ok: Set by the kernel to indicate success so far.
132  * evlogd replaces the frst_kernel_ok value with one of the following:
133  *      frst_already_registered: No need to register the facility.  It's
134  *              already registered, and the code matches.
135  *      frst_registered_ok: We registered it, with the expected results.
136  *      frst_registration_failed: We tried to register it, but couldn't.
137  *      frst_faccode_mismatch: A facility by that name is in the registry,
138  *              but its code doesn't match fr_kernel_fac_code.
139  */
140 typedef enum {
141         frst_kernel_failed = -1,
142         frst_kernel_ok = 0,
143         frst_already_registered,
144         frst_registered_ok,
145         frst_registration_failed,
146         frst_faccode_mismatch
147 } evl_facreg_rq_status_t;
148
149 struct evl_facreg_rq {
150         posix_log_facility_t    fr_kernel_fac_code;
151         posix_log_facility_t    fr_registry_fac_code;
152         evl_facreg_rq_status_t  fr_rq_status;
153 };
154
155 #ifdef __KERNEL__
156 /*
157  * Reserved Facilities
158  */
159 #define LOG_KERN        (0<<3)  /* Kernel Facility */
160 #define LOG_AUTHPRIV    (10<<3) /* security/authorization messages (private) */
161 /*
162  * priorities (these are ordered)
163  */
164 #define LOG_EMERG   0   /* system is unusable */
165 #define LOG_ALERT   1   /* action must be taken immediately */
166 #define LOG_CRIT    2   /* critical conditions */
167 #define LOG_ERR     3   /* error conditions */
168 #define LOG_WARNING 4   /* warning conditions */
169 #define LOG_NOTICE  5   /* normal but significant condition */
170 #define LOG_INFO    6   /* informational */
171 #define LOG_DEBUG   7   /* debug-level messages */
172
173 #ifdef CONFIG_EVLOG
174 extern int evl_writek(posix_log_facility_t facility, int event_type,
175                       posix_log_severity_t severity, unsigned int flags, ...);
176
177 extern int evl_vwritek(posix_log_facility_t facility, int event_type,
178                       posix_log_severity_t severity, unsigned int flags,va_list args);
179
180 extern int posix_log_printf(posix_log_facility_t facility, int event_type,
```

```
181                posix_log_severity_t severity, unsigned int flags,
182                const char *fmt, ...);
183
184 extern int posix_log_vprintf(posix_log_facility_t facility, int event_type,
185                posix_log_severity_t severity, unsigned int flags,
186                const char *fmt, va_list args);
187
188 extern int posix_log_write(posix_log_facility_t facility, int event_type,
189                posix_log_severity_t severity, const void *buf,
190                size_t len, int format, unsigned int flags);
191
192 extern int evl_gen_facility_code(const char *fname,
193                posix_log_facility_t *fcode);
194
195 extern int evl_register_facility(const char *fname,
196                posix_log_facility_t *fcode);
197 #else   /* ! CONFIG_EVLOG */
198 inline int evl_writek(posix_log_facility_t facility, int event_type,
199                posix_log_severity_t severity, unsigned int flags, ...)
200                { return -ENOSYS; }
201
202 inline int evl_vwritek(posix_log_facility_t facility, int event_type,
203                posix_log_severity_t severity, unsigned int flags,va_list args)
204                { return -ENOSYS; }
205
206 inline int posix_log_printf(posix_log_facility_t facility, int event_type,
207                posix_log_severity_t severity, unsigned int flags,
208                const char *fmt, ...)
209                { return -ENOSYS; }
210
211 inline int posix_log_vprintf(posix_log_facility_t facility, int event_type,
212                posix_log_severity_t severity, unsigned int flags,
213                const char *fmt, va_list args)
214                { return -ENOSYS; }
215
216 inline int posix_log_write(posix_log_facility_t facility, int event_type,
217                posix_log_severity_t severity, const void *buf,
218                size_t len, int format, unsigned int flags)
219                { return -ENOSYS; }
220
221 inline int evl_gen_facility_code(const char *fname,
222                posix_log_facility_t *fcode)
223                { return -ENOSYS; }
224
225 inline int evl_register_facility(const char *fname,
226                posix_log_facility_t *fcode)
227                { return -ENOSYS; }
228 #endif  /* CONFIG_EVLOG */
229 #endif  /* __KERNEL__ */
230
231 #endif
```

```
/*
 * Linux Event Logging for the Enterprise
 * Copyright (c) International Business Machines Corp., 2001
 *
 *
 * 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.
 *
 *  Please send e−mail to lkessler@users.sourceforge.net if you have
 *  questions or comments.
 *
 *  Project Website:  http://evlog.sourceforge.net/
 *
 */

#include <linux/config.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <linux/spinlock.h>
#include <linux/time.h>
#include <linux/smp.h>
#include <linux/ptrace.h>
#include <linux/string.h>
#include <linux/interrupt.h>
#include <asm/uaccess.h>
#include <asm/semaphore.h>

#ifdef __i386__
#include <asm/fixmap.h>
#include <asm/mpspec.h>
#include <asm/io_apic.h>
#include <asm/apic.h>
#endif

#include <asm/bitops.h>
#include <asm/smp.h>

#include <linux/evl_log.h>

#define EVL_BUF_SIZE (CONFIG_EVLOG_BUFSIZE * 1024)   /* EVL buffer size */
#define EVL_BUF_FREESPACE (64*1024) /* max free space reqd after buff */
                                    /* overrun to start writing events again. */
/*
 * This data structure describes the circular buffer that is written into
 * by evl_kwrite_buf() and drained by evl_kbufread().
 *
 * bf_buf, bf_len, and bf_end are the start, length, and end of the buffer,
 * and in the current implementation these remain constant.
 *
 * bf_tail advances as event records are logged to the buffer, and bf_head
 * advances as records are drained from the buffer.  bf_curr is used
 * internally by certain functions.  bf_dropped maintains a count of
 * records that have been dropped due to buffer overrun.
 */
struct cbuf {
        unsigned char   *bf_buf;        /* base buffer address */
        unsigned int    bf_len;         /* buffer length */
        unsigned int    bf_dropped;     /* (internal) dropped count */
        unsigned char   *bf_head;       /* head−pointer for circ. buf */
        unsigned char   *bf_tail;       /* tail−pointer for circ. buf */
        unsigned char   *bf_curr;       /* (internal) current write ptr */
        unsigned char   *bf_end;        /* end buffer address */
};

static unsigned char evl_buffer[EVL_BUF_SIZE + sizeof(long)];

static struct cbuf evl_ebuf = {
        evl_buffer,
        EVL_BUF_SIZE,
        0,
        evl_buffer,
        evl_buffer,
        evl_buffer,
        evl_buffer + EVL_BUF_SIZE
};

#define min(a,b) (((a)<(b))?(a):(b))

```

```
/*
 * This is for the serialisation of reads from the kernel buffer.
 */
static DECLARE_MUTEX(evl_read_sem);
DECLARE_WAIT_QUEUE_HEAD(readq);
spinlock_t ebuf_lock = SPIN_LOCK_UNLOCKED;

extern void mk_rec_header(struct posix_log_entry *rechdr,
        posix_log_facility_t facility, int event_type,
        posix_log_severity_t severity, size_t recsize,
        uint recflags, int format);

extern char *copy_attr_data(va_list, char *, char *, char *, int *, int *);

extern void evl_console_print(posix_log_facility_t facility, int event_type,
        posix_log_severity_t severity, int format, uint rec_len,
        const char * databuf);
extern int console_loglevel;
extern struct console *console_drivers;


/*
 * FUNCTION     : cbufwrap
 * Caller wants to write a chunk of size len bytes into the buffer starting at
 * bf_curr.  If bf_curr is near the end of the buffer, then we may have to
 * split the chunk so that some of it appears at the end of the buffer and
 * the rest appears at the beginning.  dont_split_hdr=1 means the chunk is
 * an entire record, and we're not allowed to split the record's header --
 * we may have to leave unused space at the end of the buffer and start the
 * record at the beginning.
 *
 * Return the location where the chunk should start, or NULL if there's
 * no room for it (i.e., it would overrun the buffer's head pointer).
 *
 * ARGUMENTS    : bf_curr - where caller wants to put record
 *              : len - total record size
 *              : dont_split_hdr - 0 if it's OK to split the header, else 1
 *
 * RETURN       : where the record should be put, or NULL if no room
 */
unsigned char *
cbufwrap(unsigned char *bf_curr, size_t len, int dont_split_hdr)
{
        unsigned char *head, *end;
        unsigned char *wrapbuf = bf_curr;

        end = bf_curr + len;
        head = evl_ebuf.bf_head;

        if (bf_curr < head && end >= head) {
                /* Insufficient free space in buffer; drop record. */
                return NULL;
        }
        if (end > evl_ebuf.bf_end) {
                /* end would be of end of buffer */
                if (dont_split_hdr
                    && bf_curr + REC_HDR_SIZE > evl_ebuf.bf_end) {
                        /* Start record at start of buffer. */
                        wrapbuf = evl_ebuf.bf_buf;
                } else {
                        /* Split record */
                        len -= evl_ebuf.bf_end - bf_curr;
                }

                end = evl_ebuf.bf_buf + len;
                if (end >= head) {
                        /* Insufficient free space in buffer; drop record. */
                        return NULL;
                }
        }
        return wrapbuf;
}

/*
 * FUNCTION     : cbufwrite
 * Copy the len bytes starting at s into the buffer starting at bufp,
 * splitting the data if we run off the end of the buffer.
 *
 * ARGUMENTS    : The pointer in cbuf to write at,
 *                the buffer to write,
 *                the number of bytes to write
 * RETURN       : The pointer in cbuf at the end of the copied data
 * NOTE
 *      bf_head is owned by the drainer. bf_curr is allowed to be equal to
 *      bf_head only when buffer is empty indicated by drainer. It is never
 *      set to be equal by writer.
 */
unsigned char *
cbufwrite(unsigned char *bufp, unsigned char *s, size_t len)
{
```

```
181         register unsigned char *e;
182         register size_t n, lw;
183
184         e = bufp + len;
185         n = len;
186
187         if (e > evl_ebuf.bf_end) {
188                 lw = evl_ebuf.bf_end − bufp;
189                 n −= lw;
190                 e = evl_ebuf.bf_buf + n;
191                 memcpy(bufp, s, lw);
192                 memcpy(evl_ebuf.bf_buf, s + lw, n);
193                 return e;
194         } else {
195                 memcpy(bufp, s, n);
196                 return (e == evl_ebuf.bf_end) ? evl_ebuf.bf_buf : e;
197         }
198 }
199
200 /*
201  * FUNCTION     : cbufptr
202  * Return cbuf_ptr + offset, taking possible wraparound into account.
203  *
204  * ARGUMENTS    : starting pointer in cbuf,
205  *                offset in bytes from pointer
206  * RETURN       : The pointer in circular buf that is offset bytes from
207  *                starting pointer.
208  */
209 unsigned char *
210 cbufptr(unsigned char *cbuf_ptr, size_t offset)
211 {
212         register unsigned char *e;
213         register size_t lw;
214
215         e = cbuf_ptr + offset;
216         if (e >= evl_ebuf.bf_end) {
217                 lw = evl_ebuf.bf_end − cbuf_ptr;
218                 e = evl_ebuf.bf_buf + (offset − lw);
219         }
220         return e;
221 }
222
223 /*
224  * FUNCTION     : copy_data_to_cbuf
225  * Copies the indicated record into the buffer at evl_ebuf.bf_curr.
226  * Assumes we've previously verified that the header won't be split.
227  * Updates evl_ebuf.bf_curr to point past the record just copied in.
228  *
229  * ARGUMENTS    : Record header
230  *                pointer to variable data
231  */
232 int
233 copy_data_to_cbuf(struct posix_log_entry *rhdr, unsigned char *vbuf)
234 {
235         memcpy(evl_ebuf.bf_curr, rhdr, REC_HDR_SIZE);
236         evl_ebuf.bf_curr += REC_HDR_SIZE;
237
238         if (rhdr->log_size > 0) {
239                 evl_ebuf.bf_curr =
240                         cbufwrite(evl_ebuf.bf_curr, vbuf, rhdr->log_size);
241         }
242
243         return 0;
244 }
245
246 /*
247  * EVL circular buffer had been full and caused later messages to be
248  * dropped. Now the buffer has space.  (Still it is better to identify the
249  * cause so it doesn't repeat. The buffer gets full if the rate of incoming
250  * messages is much higher than can be drained. This could happen if messages
251  * are repeated at an excessively high rate or the daemon in user−space is
252  * not running.)
253  *
254  * Now that we can log events again, log one giving the number of events
255  * dropped.
256  */
257 static int
258 log_dropped_recs_event(void)
259 {
260         unsigned char sbuf[255];
261         struct posix_log_entry drechdr;
262         size_t vbuflen;
263         unsigned char *oldcur = evl_ebuf.bf_curr;
264
265         snprintf(sbuf, sizeof(sbuf), "%d event records dropped due to EVL buffer overflow.",
266                 evl_ebuf.bf_dropped);
267         evl_ebuf.bf_dropped = 0;
268         vbuflen = strlen(sbuf) + 1;
269         mk_rec_header(&drechdr, LOG_KERN, EVL_BUFFER_OVERRUN, LOG_INFO,
270                 vbuflen, 0, POSIX_LOG_STRING);
```

```
271         evl_ebuf.bf_curr = cbufwrap(evl_ebuf.bf_curr, REC_HDR_SIZE+vbuflen, 1);
272         if (evl_ebuf.bf_curr != (unsigned char *)NULL) {
273                 copy_data_to_cbuf(&drechdr, sbuf);
274                 return 0;
275         } else {
276                 /*
277                  * This shouldn't happen, since EVL_BUF_FREESPACE is much
278                  * bigger than the event we're logging here.
279                  */
280                 evl_ebuf.bf_curr = oldcur;
281                 return -1;
282         }
283 }
284
285 /*
286  * FUNCTION     : evl_check_buf
287  * ARGUMENTS    : NONE
288  * RETURN       : -1 for failure, ie. insufficient buffer space
289  *                 0 for success
290  * If buffer free space is greater than the applicable water-mark,
291  * returns 0.  If not, return -1.  Sets evl_buf.bf_curr to the location
292  * where the next record should go.
293  *
294  * Once the high water mark is hit and failure is returned (discarded
295  * messages) it sets a substantial low water mark before permitting
296  * messages to be buffered again.  It counts the number of discards
297  * in the meantime and reports them when restarted.  If the water
298  * marks were equivalent, then there could be a thrashing of stops
299  * and starts, making the discarded message reporting annoying.
300  *
301  */
302 int
303 evl_check_buf(void)
304 {
305         unsigned char *head, *tail;
306         size_t water_mark, avail;
307
308         head    = evl_ebuf.bf_head;
309         tail    = evl_ebuf.bf_tail;
310         avail   = (head <= tail) ?
311               (evl_ebuf.bf_len - (tail - head)) :
312               (head - tail);
313
314         if (evl_ebuf.bf_dropped != 0) {
315                 /*
316                  * Still recovering from buffer overflow.
317                  * Apply the low water mark.
318                  */
319                 water_mark = min(EVL_BUF_FREESPACE, evl_ebuf.bf_len / 2);
320         } else {
321                 water_mark = REC_HDR_SIZE;
322         }
323
324         if (avail < water_mark) {
325                 return -1;
326         }
327
328         /* There's enough free buffer space.  Return success. */
329         evl_ebuf.bf_curr = tail;
330         if (evl_ebuf.bf_dropped != 0) {
331                 return log_dropped_recs_event();
332         }
333         return 0;
334 }
335
336 /*
337  * FUNCTION     : evl_getnext_rec
338  * ARGUMENTS    : rec is a pointer to the log event record.
339  *                The next record pointer mustn't be beyond tail.
340  * RETURN       : This function returns a pointer to the place to start the
341  *                next record in the circular buffer.  As a special case,
342  *                if rec is NULL, then the location of the first record at
343  *                or after bf_head is returned; else the record following
344  *                rec is returned.
345  */
346 unsigned char *
347 evl_getnext_rec(unsigned char *rec, size_t recsize, unsigned char *tail)
348 {
349         if (rec == NULL) {
350                 rec = evl_ebuf.bf_head;
351         } else {
352                 rec = cbufptr(rec, recsize);
353         }
354
355         if (rec == tail) {
356                 return rec;
357         }
358
359         /* Check for wrap. */
360         if ((rec + REC_HDR_SIZE) > evl_ebuf.bf_end) {
```

```
361                     rec = evl_ebuf.bf_buf;
362             }
363             return rec;
364     }
365
366     #if 0
367     /*
368      * FUNCTION     : evl_uwrite_buf − Used to write user space messages.
369      * ARGUMENTS    : buf is a pointer to buffer that needs to be written into the
370      *                        EVL kernel buffer.
371      *                      : buflen is length of the buffer to be written.
372      * RETURN       : 0 if writing to buffer is successful, else −ve value.
373      * USAGE        : This function acquires the circular buffer lock and copies the
374      *                event record and data to buffer.
375      *
376      * NOTE:                This function is left over from earlier event logging releases
377      *                      when user events were passed into the kernel buffer (instead of
378      *                      being passed directly to the evlogd daemon via socket in the
379      *                      later release).  For some environments, using this function
380      *                      could be the preferred method (and thus it was left in for now).
381      */
382
383     int
384     evl_uwrite_buf(char *buf, uint buflen)
385     {
386             int error = 0, recsize;
387             unsigned char *old_cur = NULL;
388             evl_buf_rec_t *rec;
389             char *kbuf;
390             char *oldtail = evl_ebuf.bf_tail; /* Used to wake the read call if it sleeps */
391             long iflags;
392
393             if (buflen < REC_HDR_SIZE) {
394                     return −EINVAL;
395             }
396             kbuf = kmalloc(buflen, GFP_KERNEL);
397             if (kbuf == (char *)NULL) {
398                     return −ENOMEM;
399             }
400             error = copy_from_user(kbuf, buf, buflen);
401             if (error) {
402                     kfree(kbuf);
403                     return −EINVAL;
404             }
405             rec = (evl_buf_rec_t *)kbuf;
406             if (buflen != (REC_HDR_SIZE + rec−>rechdr.log_size)) {
407                     kfree(kbuf);
408                     return −EINVAL;
409             }
410             /*
411              * An event came from user space but has pretended to be
412              * from kernel space. We just return with EPERM
413              */
414             if (rec−>rechdr.log_flags & EVL_KERNEL_EVENT) {
415                     kfree(kbuf);
416                     return −EPERM;
417             }
418             rec−>rechdr.log_processor = smp_processor_id();
419
420             spin_lock_irqsave(&ebuf_lock, iflags);
421             if (evl_check_buf() < 0) {
422                     evl_ebuf.bf_dropped++;
423                     spin_unlock_irqrestore(&ebuf_lock, iflags);
424                     kfree(kbuf);
425                     return −ENOSPC;
426             }
427
428             /* store off buf_curr to restore if necessary */
429             old_cur = evl_ebuf.bf_curr;
430
431             /*
432              * The buffer must have enough space to store the core structure
433              */
434             recsize = REC_HDR_SIZE + rec−>rechdr.log_size;
435
436             /*
437              *If insufficient space in buffer after wrap around, drop record
438              */
439             if ((evl_ebuf.bf_curr = cbufwrap(evl_ebuf.bf_curr, recsize, 1))
440                     != (char *)NULL) {
441                     copy_data_to_cbuf(&(rec−>rechdr),(char *)(kbuf + REC_HDR_SIZE));
442             }
443             if (evl_ebuf.bf_curr == (char *)NULL){
444                     error = −ENOSPC;
445                     evl_ebuf.bf_dropped++;
446                     evl_ebuf.bf_curr = old_cur;
447             } else {
448                     evl_ebuf.bf_tail = evl_ebuf.bf_curr;
449                     if ((evl_ebuf.bf_head == oldtail) &&
450                             (evl_ebuf.bf_head  != evl_ebuf.bf_tail)) {
```

```
451                             wake_up_interruptible(&readq);
452                     }
453             }
454
455             spin_unlock_irqrestore(&ebuf_lock, iflags);
456             kfree(kbuf);
457             return error;
458     }
459     #endif
460
461
462     /*
463      * FUNCTION     : evl_kbufread − Used to read event records from the EVL
464      *                circular buffer.
465      * ARGUMENTS    : retbuf is a pointer to the buffer to be filled with the
466      *                event records.
467      *              : bufsize is length of the buffer allocated by the user.
468      * RETURN       : Number of bytes copied if read is successful, else −ve value.
469      *                event record and data to buffer.
470      */
471
472     int
473     evl_kbufread(unsigned char *retbuf, size_t bufsize)
474     {
475             unsigned char *rec;
476             register size_t rec_size;
477             int error = 0;
478             int retbuflen = 0;
479             unsigned char *tail, *buf = retbuf;
480
481             /*
482              * the read request size must be at least rec_hdr_t size
483              */
484             if (bufsize < REC_HDR_SIZE) {
485                     return −EINVAL;
486             }
487             /*
488              * Serialize all reads, just in case someone got sneaky
489              */
490             error = down_interruptible(&evl_read_sem);
491             if (error == −EINTR) {
492                     return −EINTR;
493             }
494             /*
495              * Go to sleep if the buffer is empty.
496              */
497             error = wait_event_interruptible(readq,
498                     (evl_ebuf.bf_head != evl_ebuf.bf_tail));
499             if (error) {
500                     up(&evl_read_sem);
501                     return error;
502             }
503             /*
504              * Assemble message(s) into the user buffer, as many as will
505              * fit.  On running out of space in the buffer, try to copy
506              * the header for the overflowing message.  This means that
507              * there will always be at least a header returned.  The caller
508              * must compare the numbers of bytes returned (remaining) with
509              * the length of the message to see if the entire message is
510              * present.  A subsequent read will get the entire message,
511              * including the header (again).
512              */
513             tail = evl_ebuf.bf_tail;
514             rec = evl_getnext_rec(NULL, 0, tail);   /* typically evl_ebuf.bf_head */
515             if (rec == NULL) {
516                     /* Should not happen. Buffer must have atleast one record. */
517                     error = −EFAULT;
518                     goto out;
519             }
520
521             do {
522     #if defined(__ia64__)
523                     evl_buf_rec_t record;
524                     memcpy(&record, rec, sizeof(evl_buf_rec_t));
525                     rec_size = REC_HDR_SIZE + record.rechdr.log_size;
526     #else
527                     evl_buf_rec_t *p_rec;
528                     p_rec = (evl_buf_rec_t *) rec;
529                     rec_size = REC_HDR_SIZE + p_rec->rechdr.log_size;
530     #endif
531
532                     if (bufsize < REC_HDR_SIZE) {
533                             /* user buffer is smaller than header */
534                             break;
535                     }
536                     if (bufsize < rec_size) {
537                             /*
538                              * Copyout only the header 'cause user buffer can't
539                              * hold full record.
540                              */
```

```
541                             error = copy_to_user(buf, rec, REC_HDR_SIZE);
542                             if (error) {
543                                     error = −EFAULT;
544                                     break;
545                             }
546                             bufsize −= REC_HDR_SIZE;
547                             retbuflen += REC_HDR_SIZE;
548                             break;
549                     }
550                     if ((rec + rec_size) > evl_ebuf.bf_end) {
551                             size_t lw = evl_ebuf.bf_end − rec;
552                             error = copy_to_user(buf, rec, lw);
553                             if (!error) {
554                                     error = copy_to_user(buf + lw, evl_ebuf.bf_buf,
555                                             rec_size − lw);
556                             }
557                     } else {
558                             error = copy_to_user(buf, rec, rec_size);
559                     }
560                     if (error) {
561                             error = −EFAULT;
562                             break;
563                     }
564                     rec = evl_getnext_rec(rec, rec_size, tail);
565                     buf += rec_size;
566                     bufsize −= rec_size;
567                     retbuflen += rec_size;
568             } while (rec != tail);
569
570             if (error == 0) {
571                     evl_ebuf.bf_head = rec;
572                     error = retbuflen;
573             }
574
575     out:
576             up(&evl_read_sem);
577             return(error);
578     }
579
580     /*
581      * FUNCTION     : kwrite_args
582      * Called by evl_kwrite_buf() to write to the buffer an event that was
583      * created by evl_writek() (or some other function with args that must
584      * be interpreted by copy_attr_data()).
585      *
586      * RETURN       : 0 on success, −ENOSPC (or an error code from copy_attr_data())
587      *                on failure
588      *                On success evl_ebuf.bf_curr is updated to point just past
589      *                the event we just wrote to the buffer, and *pvardata points
590      *                to the variable−data portion of the record.
591      */
592     static int
593     kwrite_args(struct posix_log_entry *rec_hdr, va_list args,
594             unsigned char **pvardata)
595     {
596             unsigned char *hdr, *vardata, *rec_end;
597             int error = 0;
598             uint var_rec_len = 0;
599
600             hdr = cbufwrap(evl_ebuf.bf_curr, REC_HDR_SIZE, 1);
601             if (hdr == (unsigned char*) NULL) {
602                     return −ENOSPC;
603             }
604             vardata = hdr + REC_HDR_SIZE;
605             rec_end = copy_attr_data(args, evl_ebuf.bf_buf, evl_ebuf.bf_end,
606                     vardata, &var_rec_len, &error);
607             if (rec_end == (unsigned char*) NULL) {
608                     if (error == 0) {
609                             error = −ENOSPC;
610                     }
611                     return error;
612             }
613             rec_hdr−>log_size = var_rec_len;
614             if (var_rec_len == 0) {
615                     rec_hdr−>log_format = POSIX_LOG_NODATA;
616             } else if (var_rec_len > POSIX_LOG_ENTRY_MAXLEN) {
617                     rec_hdr−>log_size = POSIX_LOG_ENTRY_MAXLEN;
618                     rec_hdr−>log_flags |= POSIX_LOG_TRUNCATE;
619             }
620             memcpy(hdr, rec_hdr, REC_HDR_SIZE);
621             *pvardata = vardata;
622             evl_ebuf.bf_curr = rec_end;
623             return 0;
624     }
625
626     /*
627      * FUNCTION     : kwrite_buf
628      * Called by evl_kwrite_buf() to write to the buffer the event record
629      * consisting of rec_hdr and vardata.
630      *
```

```
631  * RETURN       : 0 on success, −ENOSPC on failure
632  *               On success evl_ebuf.bf_curr is updated to point just past
633  *               the event we just wrote to the buffer.
634  */
635 static int
636 kwrite_buf(struct posix_log_entry *rec_hdr, unsigned char *vardata)
637 {
638         size_t recsize;
639         unsigned char *rec;
640         if (rec_hdr->log_size > POSIX_LOG_ENTRY_MAXLEN) {
641                 rec_hdr->log_size = POSIX_LOG_ENTRY_MAXLEN;
642                 rec_hdr->log_flags |= POSIX_LOG_TRUNCATE;
643         }
644         recsize = REC_HDR_SIZE + rec_hdr->log_size;
645         rec = cbufwrap(evl_ebuf.bf_curr, recsize, 1);
646         if (rec == (unsigned char *)NULL) {
647                 return −ENOSPC;
648         }
649
650         evl_ebuf.bf_curr = rec;
651         copy_data_to_cbuf(rec_hdr, vardata);
652
653         /*
654          * If the variable data is a truncated string, make sure it
655          * ends with a null character.
656          */
657         if ((rec_hdr->log_flags & POSIX_LOG_TRUNCATE) &&
658             rec_hdr->log_format == POSIX_LOG_STRING) {
659                 if (evl_ebuf.bf_curr == evl_ebuf.bf_buf) {
660                         *(evl_ebuf.bf_end − 1) = '\0';
661                 } else {
662                         *(evl_ebuf.bf_curr − 1) = '\0';
663                 }
664         }
665         return 0;
666 }
667
668 /*
669  * FUNCTION:    log_event_to_console
670  * Called by evl_kwrite_buf() to log to the console the event record
671  * consisting of rec_hdr and vardata.
672  */
673 static void
674 log_event_to_console(const struct posix_log_entry *rec_hdr,
675         const unsigned char *vardata)
676 {
677         unsigned char consolebuf[16];
678
679         if (rec_hdr->log_format == POSIX_LOG_BINARY
680             && (evl_ebuf.bf_buf <= vardata && vardata < evl_ebuf.bf_end)) {
681                 /*
682                  * vardata points into the circular buffer.  This means
683                  * that vardata was pieced together by copy_attr_data(),
684                  * and the circular buffer is the closest thing we have
685                  * to a contiguous copy.  Since the format is BINARY, we
686                  * "know" that evl_console_print() cares only about the
687                  * first 16 bytes.
688                  */
689                 int nimportant = min(16, rec_hdr->log_size);
690                 if (vardata + nimportant > evl_ebuf.bf_end) {
691                         /*
692                          * The section of interest wraps back to the start
693                          * of the buffer.  Copy it into consolebuf to make
694                          * it contiguous.
695                          */
696                         int first_part =  evl_ebuf.bf_end − vardata;
697                         memcpy(consolebuf, vardata, first_part);
698                         memcpy(consolebuf+first_part, evl_ebuf.bf_buf,
699                                 nimportant − first_part);
700                         vardata = consolebuf;
701                 }
702         }
703         evl_console_print(rec_hdr->log_facility, rec_hdr->log_event_type,
704                 rec_hdr->log_severity, rec_hdr->log_format, rec_hdr->log_size,
705                 vardata);
706 }
707
708 /*
709  * FUNCTION     : evl_kwrite_buf − Used to write kernel level messages.
710  * RETURN       : 0 if writing to buffer is successful, else −errno.
711  */
712
713 int
714 evl_kwrite_buf(posix_log_facility_t    fac,
715                 int                    ev_type,
716                 posix_log_severity_t   sev,
717                 int                    format,
718                 unsigned char          *recbuf,
719                 uint                   var_rec_len,
720                 uint                   flags,
```

```
721                     va_list                 args)    /* Used only for evl_writek */
722 {
723         uint recflags = flags;
724         struct posix_log_entry rec_hdr;
725         int error = 0;
726         unsigned char *oldtail = evl_ebuf.bf_tail;
727                 /* Used to wake the read call if it sleeps */
728         long iflags;    /* for spin_lock_irqsave() */
729 
730         if (sev > LOG_DEBUG) {
731                 return -EINVAL;
732         }
733 
734         recflags |= EVL_KERNEL_EVENT;   /* kernel mesagge */
735         if (in_interrupt()) {
736                 recflags |= EVL_INTERRUPT;
737         }
738         mk_rec_header(&rec_hdr, fac, ev_type, sev, var_rec_len, recflags,
739                 format);
740 
741         spin_lock_irqsave(&ebuf_lock, iflags);
742         if (evl_check_buf() < 0) {
743                 evl_ebuf.bf_dropped++;
744                 spin_unlock_irqrestore(&ebuf_lock, iflags);
745                 return -ENOSPC;
746         }
747 
748         if (args) {
749                 /*
750                  * Copy the header and format the args into the circular
751                  * buffer, and make recbuf point to the variable data.
752                  */
753                 error = kwrite_args(&rec_hdr, args, &recbuf);
754                 if (error == 0
755                     && (rec_hdr.log_flags & EVL_PRINTK) == 0
756                     && sev < console_loglevel
757                     && console_drivers) {
758                         log_event_to_console(&rec_hdr, recbuf);
759                 }
760         } else {
761                 error = kwrite_buf(&rec_hdr, recbuf);
762                 /* Caller will call evl_console_print() as needed. */
763         }
764 
765         if (error == 0) {
766                 evl_ebuf.bf_tail = evl_ebuf.bf_curr;
767                 if ((evl_ebuf.bf_head == oldtail) &&
768                     (evl_ebuf.bf_head != evl_ebuf.bf_tail)) {
769                         wake_up_interruptible(&readq);
770                 }
771         } else if (error == -ENOSPC) {
772                 evl_ebuf.bf_dropped++;
773         }
774         spin_unlock_irqrestore(&ebuf_lock, iflags);
775         return error;
776 }
```

```
/*
 * Linux Event Logging for the Enterprise
 * Copyright (c) International Business Machines Corp., 2001
 *
 *
 *  This library is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU Lesser General Public
 *  License as published by the Free Software Foundation; either
 *  version 2.1 of the License, or (at your option) any later version.
 *
 *  This library 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
 *  Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this library; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 *
 *  Please send e−mail to lkessler@users.sourceforge.net if you have
 *  questions or comments.
 *
 *  Project Website:  http://evlog.sourceforge.net/
 *
 */

#ifndef _EVL_LIST_H_
#define _EVL_LIST_H_

#ifdef __cplusplus
extern "C" {
#endif

struct evl_listnode {
        struct evl_listnode     *li_next, *li_prev;
        const void              *li_data;
};
typedef struct evl_listnode evl_list_t;
typedef struct evl_listnode evl_listnode_t;

extern evl_listnode_t *_evlAllocListNode();
extern evl_listnode_t *_evlMkListNode(const void *data);
extern evl_list_t *_evlAppendToList(evl_list_t *head, const void *data);
extern int _evlGetListSize(const evl_list_t *head);
extern evl_listnode_t *_evlFindNamedItemInList(const evl_list_t *head,
        const char *name);
extern void _evlFreeList(evl_list_t *head, int freeData);
extern evl_listnode_t *_evlGetNthNode(evl_list_t *head, int n);
extern void *_evlGetNthValue(evl_list_t *head, int n);
extern void _evlInsertListNode(evl_listnode_t *insertMe,
        evl_listnode_t *beforeMe);
extern evl_list_t *_evlInsertToList(void *insertMe, evl_listnode_t *beforeMe,
        evl_list_t *list);
extern evl_list_t *_evlRemoveNode(evl_listnode_t *removeMe, evl_list_t *head,
        void (*free_data_fptr)(void *data));
extern void removeNode(evl_listnode_t * p, void (* free_data_fptr)(void *data));
extern void removeAllNodes(evl_listnode_t *head, void (*free_data_fptr)(void *data));

#ifdef __cplusplus
}
#endif

#endif /* _EVL_LIST_H_ */
```

```
/*
 * Linux Event Logging for the Enterprise
 * Copyright (c) International Business Machines Corp., 2001
 *
 *
 *  This library is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU Lesser General Public
 *  License as published by the Free Software Foundation; either
 *  version 2.1 of the License, or (at your option) any later version.
 *
 *  This library 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
 *  Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this library; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 *
 *  Please send e−mail to lkessler@users.sourceforge.net if you have
 *  questions or comments.
 *
 *  Project Website:  http://evlog.sourceforge.net/
 *
 */

#ifndef _EVLOG_H_
#define _EVLOG_H_

#include <sys/types.h>
#include <stdarg.h>
#include <string.h>

#include <linux/evl_log.h>

#ifdef __cplusplus
extern "C" {
#endif

#define EVLOGD_SERVER_SOCKET            "/var/evlog/evlogdsocket"
#define EVLOGD_EVTSERVER_SOCKET         "/var/evlog/evlogdevtsocket"
#define EVLNOTIFYD_SERVER_SOCKET        "/var/evlog/evlnotifydsocket"
#define EVLOG_CONF_SOCKET                       "/var/evlog/evlconfsoc"
#define EVLACTIOND_SERVER_SOCKET        "/var/evlog/evlactiondsocket"


/* Output formats */
#define EVL_COMPACT 0x1

typedef int evlrecsize_t;
#define SIZESIZE sizeof(evlrecsize_t)

/*
 * Log Management Event Types
 * These are event types that may be logged for facility LOG_LOGMGMT.
 *
 * The following event types are reserved, and are defined in other headers:
 * 1−5: These are the standard LOG_LOGMGMT event types defined in POSIX 1003.25
 * −− e.g., POSIX_LOG_MGMT_STARTMAINT.
 * 40: EVLOG_REGISTER_FAC −− A request from the kernel to register a facility.
 */

/* Event types 1−5 are reserved. */

#define EVLOG_FORK_FAILED       8
#define EVLOG_WRITE_PID         9
#define EVLOG_PID_EXISTS        10
#define EVLOG_SIG_ACT           11
#define EVLOG_OPEN_LOG_FAILED   12
#define EVLOG_WRITE_LOG_FAILED  13
#define EVLOG_READ_LOG_FAILED   14
#define EVLOG_SEEK_LOG_FAILED   15
#define EVLOG_MALLOC_FAILED     16
#define EVLOG_READBUF_FAILED    17

#define EVLOG_WRITE_REQF_FAILED         20
#define EVLOG_READ_REQF_FAILED          21
#define EVLOG_OPEN_REQF_FAILED          22
#define EVLOG_OPEN_SOCKET_FAILED        23
#define EVLOG_BROKEN_PIPE                       24
#define EVLOG_UID_OP_FAILED                     25
#define EVLOG_GID_OP_FAILED                     26
#define EVLOG_FAILED_APPENDTOLIST       27

#define EVLOG_PWFILE_LOOKUP_FAILED      28
#define EVLOG_GRFILE_LOOKUP_FAILED      29

#define EVLOG_LOADREG_FAILED            30
#define EVLOG_ACCESS_DENIED                     31

```

```
/* Event type 40 is reserved. */

#define NFY_ACCESS_DENIED               0xfa
#define NFY_ACCESS_GRANTED              0xac
#define NFY_MAX_CLIENTS                 0xca

#define EVLOGD_ACCESS_DENIED            NFY_ACCESS_DENIED
#define EVLOGD_ACCESS_GRANTED           NFY_ACCESS_GRANTED
#define EVLOGD_MAX_CLIENTS                      NFY_MAX_CLIENTS

/*
 * Logfile Header structure.
 */
typedef struct log_header {
        uint            log_magic;      /* Magic number indicating log file */
        long            log_version;    /* Event logging version */
        posix_log_recid_t last_recId;   /* Last recid in log: used by evlogd */
        off_t           reserved1;
        uint            log_generation; /* Changes during log maintenance */
} log_header_t;

/* log an event */
extern int evl_log_write(posix_log_facility_t facility, int event_type,
        posix_log_severity_t severity, unsigned int flags, ...);

/* format fixed portion of event record */
extern int evl_format_evrec_fixed(const struct posix_log_entry *entry,
        char *buf, size_t buflen, size_t *reqlen, const char *separator,
        size_t linelen, int fmt_flags);

/* format variable portion of event record */
extern int evl_format_evrec_variable(const struct posix_log_entry *entry,
        const void *var_buf, char *buf, size_t buflen, size_t *reqlen);

/* format event record according to user-supplied format */
extern int evl_format_evrec_sprintf(const struct posix_log_entry *entry,
        const void *var_buf, const char *format, char *buf,
        size_t buflen, size_t *reqlen);

/* compute facility code from name */
extern int evl_gen_facility_code(const char *fname,
        posix_log_facility_t *fcode);

/* add facility to registry, if needed, and get code */
extern int evl_register_facility(const char *fname,
        posix_log_facility_t *fcode);

/* For internal use... */
extern int _evlDumpBytes(const void *data, size_t nBytes, char *buf,
        size_t buflen, size_t *reqlen);
extern int _evlDumpBytesForce(const void *data, size_t nBytes, char *buf,
        size_t buflen, size_t *reqlen);
size_t _evlGetMaxDumpLen();

#ifdef __cplusplus
}
#endif

#endif          /* _EVLOG_H_ */
```

```
/*
 * Linux Event Logging for the Enterprise
 * Copyright (c) International Business Machines Corp., 2001
 *
 *
 *  This library is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU Lesser General Public
 *  License as published by the Free Software Foundation; either
 *  version 2.1 of the License, or (at your option) any later version.
 *
 *  This library 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
 *  Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this library; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 *
 *  Please send e−mail to lkessler@users.sourceforge.net if you have
 *  questions or comments.
 *
 *  Project Website:  http://evlog.sourceforge.net/
 *
 */

#ifndef _EVL_TEMPLATE_H_
#define _EVL_TEMPLATE_H_

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>
#include <wchar.h>
#include "posix_evlog.h"
#include "posix_evlsup.h"
#include "evl_list.h"

/*
 * A template contains the following sections:
 * − imports and typedefs
 * − header (with facility/event_type or struct name)
 * − attributes (const and record attributes are stored in a single list)
 * − formatting section
 *
 * Imports and typedefs are stored in global lists, because they are of file
 * scope.  The rest of the info is stored as part of struct _template
 * (= template_t = evl_template_t).
 */

#define EVL_ATTRSTR_MAXLEN (8*1024)
#define EVL_DEFAULT_EVENT_TYPE EVL_INVALID_EVENT_TYPE

#ifdef __cplusplus
extern "C" {
#endif

/*
 * An attribute (tmpl_attribute_t = evl_attribute_t) has the following
 * components:
 * − type
 * − name
 * − optional dimension or bit−field width
 * − value (provided by an initializer, if it's a const attribute)
 * − optional format
 *
 * A typedef also has most of these components, and so it is represented
 * using the same data structure, but with a flag of EVL_ATTR_TYPEDEF.
 */

/*** ATTRIBUTE TYPE ***/

typedef enum tmpl_base_type {
        TY_NONE,        /* absence of value */
        TY_CHAR,        /* signed char, actually */
        TY_UCHAR,
        TY_SHORT,
        TY_USHORT,
        TY_INT,
        TY_UINT,
        TY_LONG,
        TY_ULONG,
        TY_LONGLONG,
        TY_ULONGLONG,
        TY_FLOAT,
        TY_DOUBLE,
        TY_LDOUBLE,
        TY_STRING,
        TY_WCHAR,
```

```
91          TY_WSTRING,
92          TY_ADDRESS,
93          TY_STRUCT,
94          TY_PREFIX3,
95          TY_LIST,        /*for arrays of structs, and when parsing initializers*/
96          TY_STRUCTNAME,          /* converted to TY_STRUCT after lookup */
97          TY_TYPEDEF              /* type name better be a typedef name */
98  } tmpl_base_type_t;
99
100 /*
101  * NOTE: For an attribute of type struct, att->ta_type->u.st_template
102  * points to the struct's master template, and att->ta_value.val_struct
103  * will point to a clone when the attribute is populated.
104  */
105
106 typedef struct tmpl_data_type {
107         tmpl_base_type_t        tt_base_type;
108         union {
109                 char            *st_name;       /* Use this... */
110                 struct _template *st_template;  /* ... to find this. */
111                 char            *td_name;       /* typedef name */
112         }                       u;
113 } tmpl_data_type_t;
114
115 typedef struct tmpl_prefix3 {
116         unsigned short  p3_size;
117         unsigned char   p3_format;
118 } tmpl_prefix3_t;
119
120 /*** ATTRIBUTE DIMENSION ***/
121
122 typedef enum tmpl_dimension_type {
123         TMPL_DIM_NONE,          /* no dimension -- defaults to 1 */
124         TMPL_DIM_CONST,         /* explicit dimension in template */
125         TMPL_DIM_IMPLIED,       /* e.g., int x[] = {1,2,3}; */
126         TMPL_DIM_REST,          /* remainder of record */
127         TMPL_DIM_ATTR,          /* another attribute holds dimension; */
128                                 /*      u_attribute points to it */
129         TMPL_DIM_ATTNAME,       /* ditto, except u_attname is its name */
130         TMPL_DIM_PREFIX3,       /* 3-byte prefix holds array size */
131         TMPL_DIM_BITFIELD       /* td_dimension=nbits */
132 } tmpl_dimension_type_t;
133
134 /*
135  * Overloading "dimension" to include bit-fields is sort of bizarre.
136  * This was done mostly because the dimension info and the bit-field info
137  * occupy the same part of the attribute statement.
138  * A bit-field's width (in bits) is stored in td_dimension.
139  */
140 typedef struct tmpl_dimension {
141         tmpl_dimension_type_t   td_type;
142         int                     td_dimension;   /* explicit or computed */
143         int                     td_dimension2;  /* set when populated */
144         union {
145                 struct tmpl_attribute   *u_attribute;   /* TMPL_DIM_ATTR */
146                 struct tmpl_prefix3     u_prefix3;      /* TMPL_DIM_PREFIX3 */
147                 char    *u_attname;     /* may become TMPL_DIM_ATTR */
148         }                       u;
149 } tmpl_dimension_t;
150
151 /*** ATTRIBUTE VALUE ***/
152
153 typedef union tmpl_value {
154         long                    val_long;       /* also char, short, int */
155         unsigned long           val_ulong;      /* also uchar, ushort, uint */
156         void                    *val_address;   /* address of something */
157         double                  val_double;     /* also float */
158         long double             val_longdouble;
159         long long               val_longlong;
160         unsigned long long      val_ulonglong;
161         struct _template        *val_struct;
162         char                    *val_string;
163         wchar_t                 *val_wstring;
164         void                    *val_array;
165         evl_list_t              *val_list;      /* used during parsing */
166 } tmpl_value_t;
167
168 /*** ATTRIBUTE FORMAT ***/
169
170 /* An attribute's format specification */
171 typedef enum tmpl_att_fmt_type {
172         TMPL_AFS_DEFAULT,       /* No format specified */
173         TMPL_AFS_TBD,           /* Format specified; needs analysis */
174         TMPL_AFS_SCALAR,        /* Scalar attribute; format specified */
175         TMPL_AFS_DUMP,          /* %t */
176         TMPL_AFS_BITMAP,        /* %b */
177         TMPL_AFS_VALNM,         /* %v */
178         TMPL_AFS_ARRAY,         /* Array of scalars, no %I.  () stripped */
179         TMPL_AFS_IZARRAY,       /* Array with %I and/or %Z */
180         TMPL_AFS_CHARRSTR       /* Array of [u]char formatted with %s */
```

```
} tmpl_att_fmt_type_t;

/*
 * Value matches if ((val & bm_1bits) == bm_1bits) && ((val & bm_0bits) == 0).
 * For %v, we just test val == bm_1bits.
 */
typedef struct tmpl_bitmap {
        long    bm_1bits;       /* bit=1 -> must be 1 in matching val */
        long    bm_0bits;       /* bit=1 -> must be 0 in matching val */
        char    *bm_name;
} tmpl_bitmap_t;

typedef struct tmpl_att_format {
        tmpl_att_fmt_type_t     af_type;
        int                     af_shared;      /* shared with typedef */
        char                    *af_format;     /* TBD, SCALAR, ARRAY, IZARRAY*/
        char                    *af_delimiter;  /* appears between array
                                                 * elements when displayed */
        union {
                tmpl_bitmap_t   *u_bitmaps;     /* BITMAP, VALNM */
                char            u_izorder[3];   /* e.g., Z, IZ, IS, SI, ZI */
        }                       u;
} tmpl_att_format_t;

/*** ATTRIBUTE ***/

/* Attribute flags */
#define EVL_ATTR_EXISTS         0x1
#define EVL_ATTR_CONST          0x2
#define EVL_ATTR_BITFIELD       0x4
#define EVL_ATTR_CLONE          0x8     /* not shared with master template */
#define EVL_ATTR_TYPEDEF        0x10    /* a typedef, not an attribute */

typedef struct tmpl_attribute {
        const char              *ta_name;
        tmpl_data_type_t        *ta_type;
        tmpl_dimension_t        *ta_dimension;
        tmpl_att_format_t       ta_format;
        size_t                  ta_offset;      /* from start of record */
        tmpl_value_t            ta_value;
        unsigned int            ta_flags;
        int                     ta_index;
} tmpl_attribute_t;

typedef tmpl_attribute_t evlattribute_t;

/* User-visible info about an attribute */
typedef struct evlatt_info {
        unsigned int    att_flags;              /* ta_flags */
        const char      *att_name;              /* ta_name */
        int             att_type;               /* ta_type */
        int             att_isarray;            /* 1 if an array */
        int             att_dimfixed;           /* ta_dimension, sort of */
        int             att_dimpop;             /* ta_dimension2, sort of */
} evlatt_info_t;

typedef struct tmpl_struct_ref {
        const char      *sr_path;       /* struct's import path */
        struct _template *sr_template;  /* struct's template */
        int             sr_used;        /* 1 if used by current template */
} tmpl_struct_ref_t;

/*** TEMPLATE HEADER ***/

typedef enum tmpl_header_type {
        TMPL_TH_EVLOG,
        TMPL_TH_TRACE,
        TMPL_TH_STRUCT
} tmpl_header_type_t;

typedef struct tmpl_header {
        tmpl_header_type_t      th_type;
        const char              *th_description;
        union {
                struct {
                        posix_log_facility_t evl_facility;
                        int     evl_event_type;
                } u_evl;
                struct {
                        int     tra_major;
                        int     tra_minor;
                } u_trace;
                struct {
                        const char      *st_name;
                } u_struct;
        }                       u;
} tmpl_header_t;

/*** TEMPLATE ***/

```

```
/* Template flags */
#define TMPL_TF_POPULATED       0x1
#define TMPL_TF_FIXEDSIZE       0x2     /* not currently used */
#define TMPL_TF_ERROR           0x4
#define TMPL_TF_EATSALL         0x8     /* Contains an [_R_] attribute */
#define TMPL_TF_IMPORTED        0x10    /* Created by _evlReadTemplate() */
#define TMPL_TF_ALIGNED         0x20    /* Padded like a C struct */

/* List of evl_fmt_segment */
typedef evl_list_t tmpl_parsed_fmt_t;

/*
 * The device and inode number of a binary template file uniquely
 * identify an imported template.  We use these rather than a pathname
 * because there can be multiple paths (e.g., through symbolic links or
 * through . and ..) to the same template file.
 */
typedef struct tmpl_context {
        dev_t   tc_device;
        ino_t   tc_inode;
} tmpl_context_t;

typedef struct _template {
    /* Components of an unpopulated template */
        tmpl_header_t           tm_header;
        int                     tm_flags;
        evl_list_t              *tm_attributes;
        const char              *tm_format;     /* copied from template file */
        tmpl_parsed_fmt_t       *tm_parsed_format;
        short                   tm_alignment;   /* 1, 2, 4, etc. */
        short                   tm_minsize;     /*gcc weirdness wrt bitfields*/
    /* Additional components of a populated template */
        size_t                  tm_data_size;
        const struct posix_log_entry *tm_entry; /* fixed part of event rec */
        const void              *tm_data;       /* optional part */
        posix_log_recid_t       tm_recid;       /* cross-check with tm_entry */
    /* Master/clone information */
        struct _template        *tm_master;     /* who we're cloned from */
        int                     tm_ref_count;   /* how many clones or other
                                                 * templates reference this */
#ifdef _POSIX_THREADS
        pthread_mutex_t         tm_mutex;       /* protects ref count */
#endif
    /* Stuff needed by import fussing. */
        const char              *tm_name;       /* for lookup */
        tmpl_context_t          tm_context;     /* Unique ID */
        evl_list_t              *tm_imports;    /* Structs I import */
} template_t;

typedef template_t evltemplate_t;

/* Information about a data type.  See the table early in template.c. */
typedef struct tmpl_type_info {
        char    ti_size;        /* in bytes */
        char    ti_align;       /* from gcc's __alignof__ */
        char    ti_isScalar;
        char    ti_isInteger;
        const char      *ti_name;
        const char      *ti_format;     /* default format */
} tmpl_type_info_t;

/*
 * Stuff used in parsing...
 */
typedef struct tmpl_type_and_value {
        tmpl_base_type_t        tv_type;
        tmpl_value_t            tv_value;
} tmpl_type_and_value_t;

typedef struct tmpl_parser_context {
        FILE            *pc_errfile;    /* where to report errors */
        const char      *pc_pathname;   /* the file we're parsing */
        int             pc_lineno;      /* lex's line number */
        template_t      *pc_template;   /* the template under construction */
        const char      *pc_progname;   /* argv[0] */
        int             pc_errors;      /* errors not yet hung on a template */
} tmpl_parser_context_t;

/* Under what circumstances is this template being imported? */
#define TMPL_IMPORT_EXPLICIT 1  /* import statement or compiled template */
#define TMPL_IMPORT_IMPLICIT 2  /* ref from attribute statement */
#define TMPL_IMPORT_BINARY 3    /* ref from binary template file */

extern int _evlTmplMgmtFlags;
#define TMPL_REUSE1CLONE        0x1     /* Reuse the same clone repeatedly */
#define TMPL_LAZYDEPOP          0x2     /* Don't depopulate on release */

#define evl_getLongAttVal(att) ((att)->ta_value.val_long)
#define evl_getUlongAttVal(att) ((att)->ta_value.val_ulong)
#define evl_getDoubleAttVal(att) ((att)->ta_value.val_double)
```

```
#define evl_getLongdoubleAttVal(att) ((att)->ta_value.val_longdouble)
#define evl_getLonglongAttVal(att) ((att)->ta_value.val_longlong)
#define evl_getUlonglongAttVal(att) ((att)->ta_value.val_ulonglong)
#define evl_getStringAttVal(att) ((att)->ta_value.val_string)
#define evl_getAddressAttVal(att) ((att)->ta_value.val_address)
#define evl_getArrayAttVal(att) ((att)->ta_value.val_array)

/*
 * TODO: The macros in these next two sections don't adhere to the
 * convention that all our externally visible symbols start with evl_
 * or _evl or tmpl.  We need to hide them behind an ifdef.
 */
#define baseType(att) ((att)->ta_type->tt_base_type)
#define isConstAtt(att) (((att)->ta_flags & EVL_ATTR_CONST) != 0)
#define isClonedAtt(att) (((att)->ta_flags & EVL_ATTR_CLONE) != 0)
#define attExists(att) (((att)->ta_flags & EVL_ATTR_EXISTS) != 0)
#define isBitField(att) (((att)->ta_flags & EVL_ATTR_BITFIELD) != 0)
#define isTypedef(att) (((att)->ta_flags & EVL_ATTR_TYPEDEF) != 0)
#define isArray(att) ((att)->ta_dimension && !isBitField(att))
#define isStruct(att) (baseType(att) == TY_STRUCT)
#define isIntegerAtt(att) (_evlTmplTypeInfo[baseType(att)].ti_isInteger)

#define isFixedSizeTmpl(tmpl) (((tmpl)->tm_flags & TMPL_TF_FIXEDSIZE) != 0)
#define isPopulatedTmpl(tmpl) (((tmpl)->tm_flags & TMPL_TF_POPULATED) != 0)
#define isMasterTmpl(tmpl) ((tmpl)->tm_master == NULL)

/* template.c */
extern int evl_clonetemplate(evltemplate_t *master, evltemplate_t **clone);
extern int evl_populatetemplate(evltemplate_t *tmpl,
        const struct posix_log_entry *entry, const void *buf);
extern int evl_depopulatetemplate(evltemplate_t *tmpl);
extern int evltemplate_getatt(const evltemplate_t *tmpl,
        const char *attName, evlattribute_t **att);
extern int evltemplate_getatts(const evltemplate_t *tmpl,
        evlattribute_t *buf[], unsigned int bufatts, unsigned int *natts);
extern int evlatt_gettype(const evlattribute_t *att);
extern int evlatt_getinfo(const evlattribute_t *att, evlatt_info_t *info);
extern int evlatt_getstructtmpls(const evlattribute_t *att,
        const evltemplate_t **master, const evltemplate_t **clone);
extern int evlatt_getstructfromarray(const evlattribute_t *att, int index,
        const evltemplate_t **tmpl);

extern void _evlTmplSemanticError(const char *fmt, ...);
extern tmpl_type_info_t _evlTmplTypeInfo[];
extern void _evlTmplDprintf(const char *fmt, ...);
extern int _evlEndsWith(const char *s, const char *suffix);
extern tmpl_attribute_t *_evlTmplGetNthAttribute(template_t *t, int n);
extern template_t *_evlAllocTemplate();
extern void _evlFreeTemplate(template_t *t);
extern template_t *_evlMakeEvrecTemplate(posix_log_facility_t facility,
        int eventType, const char *description);
extern template_t *_evlMakeStructTemplate(const char *structName,
        const char *description);
extern tmpl_data_type_t *_evlTmplAllocDataType();
extern tmpl_dimension_t *_evlTmplAllocDimension();
extern tmpl_type_and_value_t *_evlTmplAllocTypeAndValue();
extern tmpl_attribute_t *_evlTmplAllocAttribute();
extern int _evlTmplAddAttribute(template_t *tmpl, int constAttr,
        tmpl_data_type_t *type, const char *name, tmpl_dimension_t *dim,
        tmpl_type_and_value_t *val, char **format);
extern void _evlTmplAddFormatSpec(template_t *tmpl, char *fmtSpec);
extern template_t *_evlCloneTemplate(template_t *t);
extern long _evlTmplGetValueOfIntegerAttribute(const tmpl_attribute_t *att);
extern tmpl_attribute_t *_evlTmplFindAttribute(const template_t *tmpl,
        const char *name);
extern void _evlTmplWrapup(template_t *t);
extern void _evlTmplIncRef(template_t *t);
extern void _evlTmplDecRef(template_t *t);

/* tmplmgmt.c */
extern int evl_parsetemplates(const char *source_filename,
        evltemplate_t *template_list[], size_t listsize, size_t *ntemplates,
        FILE *error_file, const char *prog_name);
extern int evl_writetemplates(const char *directory,
        const evltemplate_t *template_list[], size_t listsize);
extern int evl_readtemplate(posix_log_facility_t facility, int event_type,
        evltemplate_t **tmpl, int clone);
extern int evl_freetemplate(evltemplate_t *tmpl);
extern int evl_releasetemplate(evltemplate_t *tmpl);
extern int evltemplate_initmgr(int flags);

extern template_t *_evlFindStructTemplate(const char *structName);
extern tmpl_struct_ref_t *_evlFindStructRef(const char *structPath);
extern int _evlImportTemplateFromIdList(evl_list_t *list, int dotStar);
extern template_t *_evlImportTemplate(const char *primaryDir,
        const char *structPath, int purpose);
extern char *_evlGetParentDir(const char *path);
extern tmpl_struct_ref_t *_evlTmplMakeStructRef(template_t *t,
        const char *structPath);
extern char *_evlMakeDotPathFromList(evl_list_t *list);
```

```
451
452 /* tmplfmt.c */
453 extern int evlatt_getstring(const evlattribute_t *att, char *buf,
454         size_t buflen);
455 extern int evltemplate_formatrec(const evltemplate_t *t, char *buf,
456         size_t buflen);
457
458 extern evl_list_t *_evlTmplParseFormat(template_t *t, char *fmtSpec);
459 extern int _evlSpecialFormatEvrec(const struct posix_log_entry *entry,
460         const void *evBuf, const evltemplate_t *tmpl, const evl_list_t *list,
461         char *fmtBuf, size_t fmtBufLen, size_t *reqLen);
462
463 /* bvfmt.c */
464 extern tmpl_bitmap_t *_evlTmplCollectVformat(char *fmt, int vFmtOnly, char **endFmt);
465 extern tmpl_bitmap_t *_evlTmplCollectBformat(char *fmt, int bFmtOnly, char **endFmt);
466
467 /* serial.c */
468 extern int _evlWriteTemplate(const template_t *t, const char *path);
469 extern template_t *_evlReadTemplate(const char *path);
470
471 /* tmplgram.y */
472 extern tmpl_parser_context_t *_evlTmplGetParserContext();
473
474 #ifdef __cplusplus
475 }
476 #endif
477
478 #endif /* _EVL_TEMPLATE_H_ */
```

```
/*
 * Linux Event Logging for the Enterprise
 * Copyright (c) International Business Machines Corp., 2001
 *
 *
 *  This library is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU Lesser General Public
 *  License as published by the Free Software Foundation; either
 *  version 2.1 of the License, or (at your option) any later version.
 *
 *  This library 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
 *  Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this library; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 *  Please send e-mail to lkessler@users.sourceforge.net if you have
 *  questions or comments.
 *
 *  Project Website:  http://evlog.sourceforge.net/
 *
 */

#ifndef _EVLOG_UTIL_H_
#define _EVLOG_UTIL_H_

#include "posix_evlog.h"
#include "posix_evlsup.h"
#include "evl_list.h"

#ifdef __cplusplus
extern "C" {
#endif

/* The components of a printf-style conversion spec */
struct evl_parsed_format {
        char    fm_flags[10];   /* e.g., "#0" in "%#08x" */
        int     fm_width;       /* e.g., 2 in "%02d" */
        int     fm_precision;   /* e.g., 2 in "%6.2f" */
        char    fm_modifier[3]; /* e.g., "ll" in "lld" */
        char    fm_conversion;  /* e.g., 'f' in "%6.2f" */
        size_t  fm_length;      /* not counting % */
};

typedef enum evl_fmt_segment_type {
        EVL_FS_STRING,  /* Ordinary text from the template */
        EVL_FS_MEMBER,  /* Member of event record header -- e.g., %uid% */
        EVL_FS_ATTNAME, /* Name of an as-yet unknown attribute */
        EVL_FS_ATTR,    /* Pointer to non-standard attribute */
        EVL_FS_ERROR    /* Used to indicate a bad attribute spec */
} evl_fmt_segment_type_t;

/* One segment of a parsed format string */
typedef struct evl_fmt_segment {
        evl_fmt_segment_type_t  fs_type;
        char                    *fs_userfmt;    /* format specified by user */
        int                     fs_stringfmt;   /* 1 for %s */
        union {
                char    *fs_string;     /* EVL_FS_STRING or EVL_FS_ERROR */
                char    *fs_attname;    /* EVL_FS_ATTNAME */
        } u;
        union {
                int                     fs_member;      /* EVL_FS_MEMBER */
                struct tmpl_attribute   *fs_attribute;  /* EVL_FS_ATTR */
        } u2;
} evl_fmt_segment_t;

/*
 * This object defines a buffer that holds the result of formatting a template
 * or an individual attribute.  fb_next starts at fb_buf, and advances through
 * the buffer as strings are written to it -- usually via bprintf().
 */
typedef struct evl_fmt_buf {
        char    *fb_buf;        /* start of buffer */
        char    *fb_next;       /* next value goes here */
        char    *fb_end;        /* fb_buf + bufsz */
        int     fb_oflow;       /* 1 if we would have written past the end */
        char    fb_dummy[1];    /* used when caller gives us a null buffer */
} evl_fmt_buf_t;

/* format.c */
extern int _evlParseFmtConvSpec(const char *fmt, struct evl_parsed_format *pf);
extern int _evlValidateAttrFmt(const char *fmt, struct evl_parsed_format *pf,
        int maxWidth);
extern evl_fmt_segment_t *_evlAllocFormatSegment();
extern void _evlFreeParsedFormat(evl_list_t *pf);
extern evl_list_t *_evlParseFormat(char *fmt, int allowNonStdAtts,
```

```
91          char **errMsg);
92  extern char *_evlAllocateFmtBuf(const evl_list_t *head, size_t *bufsz);
93  extern void _evlSprintfMember(evl_fmt_buf_t *s, const char *fmt, int member,
94          const struct posix_log_entry *entry);
95
96  /* fmtbuf.c */
97  extern evl_fmt_buf_t *_evlMakeFmtBuf(char *buf, size_t bufsz);
98  extern void _evlFreeFmtBuf(evl_fmt_buf_t *f);
99  extern void _evlBprintf(evl_fmt_buf_t *f, const char *fmt, ...);
100 extern void _evlDumpBytesToFmtBuf(const void *buf, size_t nBytes,
101         evl_fmt_buf_t *f);
102
103 /* facreg.c */
104 extern posix_log_facility_t _evlGetFacCodeByCIName(const char *name,
105         _evlFacilityRegistry *facReg);
106 extern const char * _evlGetFacNameByCode(posix_log_facility_t facNum,
107         _evlFacilityRegistry *facReg);
108 extern void _evlFreeFacReg(_evlFacilityRegistry *facReg);
109
110 #ifdef __cplusplus
111 }
112 #endif
113
114 #endif /* _EVLOG_UTIL_H_ */
```

```
/*
 * Linux Event Logging for the Enterprise
 * Copyright (c) International Business Machines Corp., 2001
 *
 *
 *  This library is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU Lesser General Public
 *  License as published by the Free Software Foundation; either
 *  version 2.1 of the License, or (at your option) any later version.
 *
 *  This library 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
 *  Lesser General Public License for more details.
 *
 *  You should have received a copy of the GNU Lesser General Public
 *  License along with this library; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA
 *
 *  Please send e−mail to lkessler@users.sourceforge.net if you have
 *  questions or comments.
 *
 *  Project Website:  http://evlog.sourceforge.net/
 *
 */

#define LOG_LOGMGMT (12<<3) /* this facility defined by POSIX, but not syslog */
#ifndef _POSIX_EVLOG_H_
#define _POSIX_EVLOG_H_

#include <sys/types.h>
/* For sigevent */
#include <signal.h>
/* For facility (e.g., LOG_KERN) and severity (e.g., LOG_EMERG) codes */
#include <syslog.h>
/* For va_list */
#include <stdarg.h>

#ifdef __cplusplus
extern "C" {
#endif  /* __cplusplus */

#ifdef _POSIX_THREADS
#include <pthread.h>
#endif
#include <linux/evl_log.h>

/* Treat a log descriptor like a file descriptor for now. */
typedef struct _log_description *posix_logd_t;

/* Member selectors for posix_log_entry, for use by posix_log_memtostr  */
#define POSIX_LOG_ENTRY_DATA    0       /* variable−length data */
#define POSIX_LOG_ENTRY_RECID   1
#define POSIX_LOG_ENTRY_SIZE    2
#define POSIX_LOG_ENTRY_FORMAT  3
#define POSIX_LOG_ENTRY_EVENT_TYPE      4
#define POSIX_LOG_ENTRY_FACILITY        5
#define POSIX_LOG_ENTRY_SEVERITY        6
#define POSIX_LOG_ENTRY_UID             7
#define POSIX_LOG_ENTRY_GID             8
#define POSIX_LOG_ENTRY_PID             9
#define POSIX_LOG_ENTRY_PGRP    10
#define POSIX_LOG_ENTRY_TIME    11
#define POSIX_LOG_ENTRY_FLAGS   12
#define POSIX_LOG_ENTRY_THREAD  13
#define POSIX_LOG_ENTRY_PROCESSOR       14

/* Query purpose flags */
#define POSIX_LOG_PRPS_NOTIFY   0x1
#define POSIX_LOG_PRPS_GENERAL  0x2
#define POSIX_LOG_PRPS_SEEK     (POSIX_LOG_PRPS_GENERAL)
#define EVL_PRPS_TEMPLATE       0x4
#define EVL_PRPS_RESTRICTED     0x8
#define POSIX_LOG_PRPS_MASK     0xf

struct node;

typedef struct _posix_log_query {
        int             qu_purpose;     /* NOTIFY, SEEK, etc. */
        char            *qu_expr;       /* Saved copy of the query expr */
        struct node     *qu_tree;       /* Parse tree */
        struct evl_nonStdAtts   *qu_nonStdAtts; /* Non−standard attributes. */
} posix_log_query_t;


#define POSIX_LOG_ONCE_ONLY             0x01
#define POSIX_LOG_SEND_RECID            0x02
#define POSIX_LOG_SEND_SIGVAL           0x04
#define POSIX_LOG_NFY_MASK              0x07

```

```
#define POSIX_LOG_NFY_DISABLED          0x08
#define POSIX_LOG_ACTION_PERSIST        0x16

#define POSIX_LOG_NFY_PENDING_REMOVE    0x512

#define SI_EVLOG        SI_QUEUE
typedef size_t posix_log_notify_t;
/*
 * These are the types of messages that can be sent between the client (a
 * process that has called posix_log_notify_add_ex()) and the server (the
 * notification daemon).
 */
typedef enum _nfyMsgType {
        nmtNewRequest,                  /* new notification request */
        nmtGetRequestStatus,    /* get status of notification request */
        nmtRmRequest,                   /* remove notification request */
        nmtRequestStatus                /* response to client requests */
} notifyMsgType_t;

/*
 * The status of a notification request.
 */
typedef enum _nfyRqStatus {
        nrsEnabled,             /* Request has been accepted by server. */
        nrsDisabled,    /* Once-only request has been satisfied */
                                        /* by the server, but not removed by the */
                                        /* client. */
        nrsRemoved,             /* Request has been removed at client's request. */
        nrsNoRequest    /* Request not found by server */
} nfyRqStatus_t;

/*
 * The header of a message between client and server.
 */
typedef struct _nfyMsgHdr {
        int     nhMsgType;
        posix_log_notify_t      nhReqHandle;
        int     nhStatus;
} nfyMsgHdr_t;

typedef struct _nfyNewRqMsg {
        nfyMsgHdr_t     nwHeader;
        size_t          nwQueryLength;
        struct sigevent nwSigevent;
        int             nwFlags;
} nfyNewRqMsg_t;

#define EVL_NFY_MSGMAXSIZE (sizeof(nfyNewRqMsg_t))

/* Log-management event-type codes */
#define POSIX_LOG_MGMT_TIMEMARK 1
#define POSIX_LOG_MGMT_STARTMAINT       2
#define POSIX_LOG_MGMT_ENDMAINT 3
#define POSIX_LOG_MGMT_STOPLOGGING      4
#define POSIX_LOG_MGMT_STARTLOGGING     5

/* Per-process maximum number of concurrent open log descriptors */
#define POSIX_LOG_OPEN_MAX      100

/* Per-process max number of concurrently registered notification requests */
#define POSIX_LOG_NOTIFY_MAX    100

/* Seek directions */
#define POSIX_LOG_SEEK_START    1
#define POSIX_LOG_SEEK_END              2
#define POSIX_LOG_SEEK_FORWARD  3
#define POSIX_LOG_SEEK_BACKWARD 4
#define POSIX_LOG_SEEK_FIRST    5
#define POSIX_LOG_SEEK_LAST             6

/* Maximum length for a string returned by posix_log_memtostr */
#define POSIX_LOG_MEMSTR_MAXLEN 128

/* Log an event */
extern int posix_log_write(posix_log_facility_t facility,
        int event_type, posix_log_severity_t severity,
        const void *buf, size_t len, int format, unsigned int flags);

extern int posix_log_vprintf(posix_log_facility_t facility,
        int event_type, posix_log_severity_t severity, unsigned int flags,
        const char *format, va_list args);

extern int posix_log_printf(posix_log_facility_t facility,
        int event_type, posix_log_severity_t severity, unsigned int flags,
        const char *format, ...);

/* Open an event log for read access */
extern int posix_log_open(posix_logd_t *logdes, const char *path);

/* Read from event log */
```

```
181 extern int posix_log_read(posix_logd_t logdes, struct posix_log_entry *entry,
182         void *log_buf, size_t log_len);
183
184 /* Register for notification about new events */
185 extern int posix_log_notify_add(const posix_log_query_t *query,
186         const struct sigevent *notification, int flags,
187         posix_log_notify_t *nfyhandle);
188
189 extern int posix_log_sigval_getrecid(const union sigval sval,
190         posix_log_recid_t *recid);
191
192 extern int posix_log_siginfo_getrecid(const siginfo_t *info,
193         void *context, posix_log_recid_t *recid);
194
195 extern int posix_log_notify_get(posix_log_notify_t nfyhandle,
196         struct sigevent *notification, int *flags,
197         char *qsbuf, size_t qslen, size_t *reqlen);
198
199 /* Remove notification request */
200 extern int posix_log_notify_remove(posix_log_notify_t nfyhandle);
201
202 /* Close event log */
203 extern int posix_log_close(posix_logd_t logdes);
204
205 /* Reposition the read pointer */
206 extern int posix_log_seek(posix_logd_t logdes, const posix_log_query_t *query,
207         int direction);
208
209 /* Compare severities */
210 extern int posix_log_severity_compare(int *order, posix_log_severity_t s1,
211         posix_log_severity_t s2);
212
213 /* Create log query */
214 extern int posix_log_query_create(const char *query_string, int purpose,
215         posix_log_query_t *query, char *errbuf, size_t errlen);
216
217 extern int posix_log_query_get(const posix_log_query_t *query, int *purpose,
218         char *qsbuf, size_t qslen, size_t *reqlen);
219
220 /* Destroy log query */
221 extern int posix_log_query_destroy(posix_log_query_t *query);
222
223 /* Obtain the string equivalent of an event−record member */
224 extern int posix_log_memtostr(int member_selector,
225         const struct posix_log_entry *entry,
226         char *buf, size_t buflen);
227
228 /* Convert string to facility or facility to string */
229 extern int posix_log_factostr(posix_log_facility_t fac, char *buf,
230         size_t buflen);
231
232 extern int posix_log_strtofac(const char *str, posix_log_facility_t *fac);
233
234 #ifdef __cplusplus
235 }
236 #endif  /* __cplusplus */
237 #endif  /* _POSIX_EVLOG_H_ */
```

Table of Content