

Nov 05, 14 14:51

signal_related.txt

Page 1/14

```

1 ***** include/signal.h *****
2 /* The <signal.h> header defines all the ANSI and POSIX signals.
3  * MINIX supports all the signals required by POSIX. They are defined below.
4  * Some additional signals are also supported.
5  */
6
7 #ifndef _SIGNAL_H
8 #define _SIGNAL_H
9
10 #ifndef _ANSI_H
11 #include <ansi.h>
12 #endif
13 #ifdef _POSIX_SOURCE
14 #ifndef _TYPES_H
15 #include <sys/types.h>
16 #endif
17 #endif
18
19 /* Here are types that are closely associated with signal handling. */
20 typedef int sig_atomic_t;
21
22 #ifdef _POSIX_SOURCE
23 #ifndef _SIGSET_T
24 #define _SIGSET_T
25 typedef unsigned long sigset_t;
26 #endif
27 #endif
28
29 #define SIGHUP          1      /* hangup */
30 #define SIGINT          2      /* interrupt (DEL) */
31 #define SIGQUIT         3      /* quit (ASCII FS) */
32 #define SIGILL          4      /* illegal instruction */
33 #define SIGTRAP         5      /* trace trap (not reset when caught) */
34 #define SIGABRT         6      /* IOT instruction */
35 #define SIGIOT          6      /* SIGABRT for people who speak PDP-11 */
36 #define SIGUNUSED       7      /* spare code */
37 #define SIGFPE          8      /* floating point exception */
38 #define SIGKILL          9      /* kill (cannot be caught or ignored) */
39 #define SIGUSR1         10     /* user defined signal # 1 */
40 #define SIGSEGV         11     /* segmentation violation */
41 #define SIGUSR2         12     /* user defined signal # 2 */
42 #define SIGPIPE         13     /* write on a pipe with no one to read it */
43 #define SIGALRM         14     /* alarm clock */
44 #define SIGTERM         15     /* software termination signal from kill */
45 #define SIGCHLD         17     /* child process terminated or stopped */
46
47 #define SIGEMT          7      /* obsolete */
48 #define SIGBUS          10     /* obsolete */
49
50 /* MINIX specific signals. These signals are not used by user proceses,
51  * but meant to inform system processes, like the PM, about system events.
52  */
53 #define SIGKMSG         18     /* new kernel message */
54 #define SIGKSIG         19     /* kernel signal pending */
55 #define SIGKSTOP        20     /* kernel shutting down */
56
57 /* Regular signals. */
58 #define SIGWINCH        21     /* window size has changed */
59
60 #define _NSIG           21     /* number of signals used */
61
62 /* POSIX requires the following signals to be defined, even if they are
63  * not supported. Here are the definitions, but they are not supported.
64  */
65 #define SIGCONT         18     /* continue if stopped */
66 #define SIGSTOP         19     /* stop signal */
67 #define SIGTSTP         20     /* interactive stop signal */
68 #define SIGTTIN         21     /* background process wants to read */
69 #define SIGTTOU         22     /* background process wants to write */
70
71 /* The sighandler_t type is not allowed unless _POSIX_SOURCE is defined. */
72 typedef void _PROTOTYPE( (__sighandler_t), (int) );
73

```

Nov 05, 14 14:51

signal_related.txt

Page 2/14

```

74 /* Macros used as function pointers. */
75 #define SIG_ERR         ((__sighandler_t) -1)      /* error return */
76 #define SIG_DFL         ((__sighandler_t) 0)      /* default signal handling */
77 #define SIG_IGN         ((__sighandler_t) 1)      /* ignore signal */
78 #define SIG_HOLD        ((__sighandler_t) 2)      /* block signal */
79 #define SIG_CATCH        ((__sighandler_t) 3)     /* catch signal */
80 #define SIG_MESS         ((__sighandler_t) 4)     /* pass as message (MINIX) */
81
82 #ifdef _POSIX_SOURCE
83 struct sigaction {
84     __sighandler_t sa_handler; /* SIG_DFL, SIG_IGN, or pointer to function */
85     sigset_t sa_mask;         /* signals to be blocked during handler */
86     int sa_flags;             /* special flags */
87 };
88
89 /* Fields for sa_flags. */
90 #define SA_ONSTACK       0x0001 /* deliver signal on alternate stack */
91 #define SA_RESETHAND     0x0002 /* reset signal handler when signal caught */
92 #define SA_NODEFER       0x0004 /* don't block signal while catching it */
93 #define SA_RESTART       0x0008 /* automatic system call restart */
94 #define SA_SIGINFO       0x0010 /* extended signal handling */
95 #define SA_NOCLDWAIT     0x0020 /* don't create zombies */
96 #define SA_NOCLDSTOP     0x0040 /* don't receive SIGCHLD when child stops */
97
98 /* POSIX requires these values for use with sigprocmask(2). */
99 #define SIG_BLOCK        0      /* for blocking signals */
100 #define SIG_UNBLOCK      1     /* for unblocking signals */
101 #define SIG_SETMASK      2     /* for setting the signal mask */
102 #define SIG_INQUIRE     4     /* for internal use only */
103 #endif /* _POSIX_SOURCE */
104
105 /* POSIX and ANSI function prototypes. */
106 _PROTOTYPE( int raise, (int _sig) );
107 _PROTOTYPE( __sighandler_t signal, (int _sig, __sighandler_t _func) );
108
109 #ifdef _POSIX_SOURCE
110 _PROTOTYPE( int kill, (pid_t _pid, int _sig) );
111 _PROTOTYPE( int sigaction,
112             (int _sig, const struct sigaction *_act, struct sigaction *_oact) );
113 _PROTOTYPE( int sigaddset, (sigset_t *_set, int _sig) );
114 _PROTOTYPE( int sigdelset, (sigset_t *_set, int _sig) );
115 _PROTOTYPE( int sigemptyset, (sigset_t *_set) );
116 _PROTOTYPE( int sigfillset, (sigset_t *_set) );
117 _PROTOTYPE( int sigismember, (const sigset_t *_set, int _sig) );
118 _PROTOTYPE( int sigpending, (sigset_t *_set) );
119 _PROTOTYPE( int sigprocmask,
120             (int _how, const sigset_t *_set, sigset_t *_oset) );
121 _PROTOTYPE( int sigsuspend, (const sigset_t *_sigmask) );
122 #endif
123
124 #endif /* _SIGNAL_H */
125

```

Nov 05, 14 14:51

signal_related.txt

Page 3/14

```

126 ***** include/sys/sigcontext.h *****
127 #ifndef _SIGCONTEXT_H
128 #define _SIGCONTEXT_H
129
130 /* The sigcontext structure is used by the sigreturn(2) system call.
131  * sigreturn() is seldom called by user programs, but it is used internally
132  * by the signal catching mechanism.
133  */
134
135 #ifndef _ANSI_H
136 #include <ansi.h>
137 #endif
138
139 #ifndef _MINIX_SYS_CONFIG_H
140 #include <minix/sys_config.h>
141 #endif
142
143 #if !defined(_MINIX_CHIP)
144 #include "error, configuration is not known"
145 #endif
146
147 /* The following structure should match the stackframe_s structure used
148  * by the kernel's context switching code. Floating point registers should
149  * be added in a different struct.
150  */
151 #if (_MINIX_CHIP == _CHIP_INTEL)
152 struct sigregs {
153     #if _WORD_SIZE == 4
154         short sr_gs;
155         short sr_fs;
156     #endif /* _WORD_SIZE == 4 */
157     short sr_es;
158     short sr_ds;
159     int sr_di;
160     int sr_si;
161     int sr_bp;
162     int sr_st; /* stack top -- used in kernel */
163     int sr_bx;
164     int sr_dx;
165     int sr_cx;
166     int sr_retreg;
167     int sr_retadr; /* return address to caller of save -- used
168                    * in kernel */
169     int sr_pc;
170     int sr_cs;
171     int sr_psw;
172     int sr_sp;
173     int sr_ss;
174 };
175
176 struct sigframe { /* stack frame created for signalled process */
177     _PROTOTYPE( void (*sf_retadr), (void) );
178     int sf_signo;
179     int sf_code;
180     struct sigcontext *sf_scp;
181     int sf_fp;
182     _PROTOTYPE( void (*sf_retadr2), (void) );
183     struct sigcontext *sf_scpcopy;
184 };
185
186 #else
187 #include "error, _MINIX_CHIP is not supported"
188 #endif
189 #endif /* _MINIX_CHIP == _CHIP_INTEL */
190
191 struct sigcontext {
192     int sc_flags; /* sigstack state to restore */
193     long sc_mask; /* signal mask to restore */
194     struct sigregs sc_regs; /* register set to restore */
195 };
196
197 #if (_MINIX_CHIP == _CHIP_INTEL)
198 #if _WORD_SIZE == 4

```

Nov 05, 14 14:51

signal_related.txt

Page 4/14

```

199 #define sc_gs sc_regs.sr_gs
200 #define sc_fs sc_regs.sr_fs
201 #endif /* _WORD_SIZE == 4 */
202 #define sc_es sc_regs.sr_es
203 #define sc_ds sc_regs.sr_ds
204 #define sc_di sc_regs.sr_di
205 #define sc_si sc_regs.sr_si
206 #define sc_fp sc_regs.sr_bp
207 #define sc_st sc_regs.sr_st /* stack top -- used in kernel */
208 #define sc_bx sc_regs.sr_bx
209 #define sc_dx sc_regs.sr_dx
210 #define sc_cx sc_regs.sr_cx
211 #define sc_retreg sc_regs.sr_retreg
212 #define sc_retadr sc_regs.sr_retadr /* return address to caller of
213                                     save -- used in kernel */
214 #define sc_pc sc_regs.sr_pc
215 #define sc_cs sc_regs.sr_cs
216 #define sc_psw sc_regs.sr_psw
217 #define sc_sp sc_regs.sr_sp
218 #define sc_ss sc_regs.sr_ss
219 #endif /* _MINIX_CHIP == _CHIP_INTEL */
220
221 /* Values for sc_flags. Must agree with <minix/jmp_buf.h>. */
222 #define SC_SIGCONTEXT 2 /* nonzero when signal context is included */
223 #define SC_NOREGLOCALS 4 /* nonzero when registers are not to be
224                           saved and restored */
225
226 _PROTOTYPE( int sigreturn, (struct sigcontext *scp) );
227
228 #endif /* _SIGCONTEXT_H */
229

```

Nov 05, 14 14:51

signal_related.txt

Page 5/14

```

230 ***** lib/posix/_sigaction.c *****
231 #include <lib.h>
232 #define sigaction _sigaction
233 #include <sys/sigcontext.h>
234 #include <signal.h>
235
236 _PROTOTYPE(int __sigreturn, (void));
237
238 PUBLIC int sigaction(sig, act, oact)
239 int sig;
240 _CONST struct sigaction *act;
241 struct sigaction *oact;
242 {
243     message m;
244
245     m.m1_i2 = sig;
246
247     /* XXX - yet more type puns because message struct is short of types. */
248     m.m1_p1 = (char *) act;
249     m.m1_p2 = (char *) oact;
250     m.m1_p3 = (char *) __sigreturn;
251
252     return(_syscall(MM, SIGACTION, &m));
253 }
254
255
256
257 ***** lib/posix/_sigprocmask.c *****
258 #include <lib.h>
259 #define sigprocmask _sigprocmask
260 #include <signal.h>
261
262
263 PUBLIC int sigprocmask(how, set, oset)
264 int how;
265 _CONST sigset_t *set;
266 sigset_t *oset;
267 {
268     message m;
269
270     if (set == (sigset_t *) NULL) {
271         m.m2_i1 = SIG_INQUIRE;
272         m.m2_l1 = 0;
273     } else {
274         m.m2_i1 = how;
275         m.m2_l1 = (long) *set;
276     }
277     if (_syscall(MM, SIGPROCMAK, &m) < 0) return(-1);
278     if (oset != (sigset_t *) NULL) *oset = (sigset_t) (m.m2_l1);
279     return(m.m_type);
280 }
281
282
283
284
285
286 ***** lib/posix/_sigsuspend.c *****
287 #include <lib.h>
288 #define sigsuspend _sigsuspend
289 #include <signal.h>
290
291 PUBLIC int sigsuspend(set)
292 _CONST sigset_t *set;
293 {
294     message m;
295
296     m.m2_l1 = (long) *set;
297     return(_syscall(MM, SIGSUSPEND, &m));
298 }
299
300

```

Nov 05, 14 14:51

signal_related.txt

Page 6/14

```

301 ***** lib/posix/_sigreturn.c *****
302 #include <lib.h>
303 #define sigfillset _sigfillset
304 #define sigjmp _sigjmp
305 #define sigprocmask _sigprocmask
306 #define sigreturn _sigreturn
307 #include <sys/sigcontext.h>
308 #include <setjmp.h>
309 #include <signal.h>
310
311 _PROTOTYPE( int sigjmp, (jmp_buf jb, int retval));
312
313 #if (_SETJMP_SAVES_REGS == 0)
314 /* 'sigreturn' using a short format jmp_buf (no registers saved). */
315 PUBLIC int sigjmp(jb, retval)
316 jmp_buf jb;
317 int retval;
318 {
319     struct sigcontext sc;
320
321     sc.sc_flags = jb[0].__flags;
322     sc.sc_mask = jb[0].__mask;
323
324     #if (CHIP == INTEL)
325         sc.sc_pc = (int) jb[0].__pc;
326         sc.sc_sp = (int) jb[0].__sp;
327         sc.sc_fp = (int) jb[0].__lb;
328     #endif
329
330     sc.sc_retreg = retval;
331     return sigreturn(&sc);
332 }
333 #endif
334
335 PUBLIC int sigreturn(scp)
336 register struct sigcontext *scp;
337 {
338     sigset_t set;
339
340     /* The message can't be on the stack, because the stack will vanish out
341     * from under us. The send part of sendrec will succeed, but when
342     * a message is sent to restart the current process, who knows what will
343     * be in the place formerly occupied by the message?
344     */
345     static message m;
346
347     /* Protect against race conditions by blocking all interrupts. */
348     sigfillset(&set); /* splhi */
349     sigprocmask(SIG_SETMASK, &set, (sigset_t *) NULL);
350
351     m.m2_l1 = scp->sc_mask;
352     m.m2_i2 = scp->sc_flags;
353     m.m2_p1 = (char *) scp;
354     return(_syscall(MM, SIGRETURN, &m)); /* normally this doesn't return */
355 }
356
357
358
359 ***** lib/i386/rts/_sigreturn.s *****
360 ! This routine is the low-level code for returning from signals.
361 ! It calls __sigreturn, which is the normal "system call" routine.
362 ! Both __sigreturn and _sigreturn are needed.
363 .sect .text; .sect .rom; .sect .data; .sect .bss
364 .sect .text
365 .define __sigreturn
366 .extern _sigreturn
367 __sigreturn:
368     add esp, 16
369     jmp __sigreturn
370
371

```

Nov 05, 14 14:51

signal_related.txt

Page 7/14

```

372 ***** lib/posix/_sigset.c *****
373 #include <lib.h>
374 /* XXX - these have to be hidden because signal() uses them and signal() is
375  * ANSI and not POSIX. It would be surely be better to use macros for the
376  * library and system uses, and perhaps macros as well as functions for the
377  * POSIX user interface. The macros would not need underlines. It may be
378  * inconvenient to match the exact semantics of the current functions
379  * because the interface is bloated by reporting errors. For library and
380  * system uses, the signal number is mostly already known to be valid
381  * before the sigset-changing routines are called.
382  */
383 #define sigaddset      _sigaddset
384 #define sigdelset      _sigdelset
385 #define sigemptyset    _sigemptyset
386 #define sigfillset     _sigfillset
387 #define sigismember     _sigismember
388 #include <signal.h>
389
390 /* Low bit of signal masks. */
391 #define SIGBIT_0      ((sigset_t) 1)
392
393 /* Mask of valid signals (0 - _NSIG). */
394 #define SIGMASK      (((SIGBIT_0 << _NSIG) << 1) - 1)
395
396 #define sigisvalid(signo) ((unsigned) (signo) <= _NSIG)
397
398 PUBLIC int sigaddset(set, signo)
399 sigset_t *set;
400 int signo;
401 {
402     if (!sigisvalid(signo)) {
403         errno = EINVAL;
404         return -1;
405     }
406     *set |= SIGBIT_0 << signo;
407     return 0;
408 }
409
410 PUBLIC int sigdelset(set, signo)
411 sigset_t *set;
412 int signo;
413 {
414     if (!sigisvalid(signo)) {
415         errno = EINVAL;
416         return -1;
417     }
418     *set &= ~(SIGBIT_0 << signo);
419     return 0;
420 }
421
422 PUBLIC int sigemptyset(set)
423 sigset_t *set;
424 {
425     *set = 0;
426     return 0;
427 }
428
429 PUBLIC int sigfillset(set)
430 sigset_t *set;
431 {
432     *set = SIGMASK;
433     return 0;
434 }
435
436 PUBLIC int sigismember(set, signo)
437 _CONST sigset_t *set;
438 int signo;
439 {
440     if (!sigisvalid(signo)) {
441         errno = EINVAL;
442         return -1;
443     }
444     if (*set & (SIGBIT_0 << signo))

```

Nov 05, 14 14:51

signal_related.txt

Page 8/14

```

445         return 1;
446     return 0;
447 }
448
449
450
451
452
453
454 ***** lib/posix/_sigpending.c *****
455 #include <lib.h>
456 #define sigpending _sigpending
457 #include <signal.h>
458
459 PUBLIC int sigpending(set)
460 sigset_t *set;
461 {
462     message m;
463
464     if (_syscall(MM, SIGPENDING, &m) < 0) return(-1);
465     *set = (sigset_t) m.m2_l1;
466     return(m.m_type);
467 }
468
469
470

```

Nov 05, 14 14:51

signal_related.txt

Page 9/14

```

471 ***** kernel/system/do_sigsend.c *****
472 /* The kernel call that is implemented in this file:
473 *   m_type:   SYS_SIGSEND
474 *
475 * The parameters for this kernel call are:
476 *   m2_i1:   SIG_PROC      # process to call signal handler
477 *   m2_p1:   SIG_CTXT_PTR  # pointer to sigcontext structure
478 *   m2_i3:   SIG_FLAGS     # flags for S_SIGRETURN call
479 *
480 */
481
482 #include "../system.h"
483 #include <signal.h>
484 #include <string.h>
485 #include <sys/sigcontext.h>
486
487 #if USE_SIGSEND
488
489 /*=====
490 *                               do_sigsend                               *
491 *=====*/
492 PUBLIC int do_sigsend(m_ptr)
493 message *m_ptr;                /* pointer to request message */
494 {
495 /* Handle sys_sigsend, POSIX-style signal handling. */
496
497 struct sigmsg smsg;
498 register struct proc *rp;
499 phys_bytes src_phys, dst_phys;
500 struct sigcontext sc, *scp;
501 struct sigframe fr, *frp;
502
503 if (!isokprocn(m_ptr->SIG_PROC)) return(EINVAL);
504 if (iskerneln(m_ptr->SIG_PROC)) return(EPERM);
505 rp = proc_addr(m_ptr->SIG_PROC);
506
507 /* Get the sigmsg structure into our address space. */
508 src_phys = umap_local(proc_addr(PM_PROC_NR), D, (vir_bytes)
509 m_ptr->SIG_CTXT_PTR, (vir_bytes) sizeof(struct sigmsg));
510 if (src_phys == 0) return(EFAULT);
511 phys_copy(src_phys, vir2phys(&smsg), (phys_bytes) sizeof(struct sigmsg));
512
513 /* Compute the user stack pointer where sigcontext will be stored. */
514 scp = (struct sigcontext *) smsg.sm_stkptr - 1;
515
516 /* Copy the registers to the sigcontext structure. */
517 memcpy(&sc.sc_regs, (char *) &rp->p_reg, sizeof(struct sigregs));
518
519 /* Finish the sigcontext initialization. */
520 sc.sc_flags = SC_SIGCONTEXT;
521 sc.sc_mask = smsg.sm_mask;
522
523 /* Copy the sigcontext structure to the user's stack. */
524 dst_phys = umap_local(rp, D, (vir_bytes) scp,
525 (vir_bytes) sizeof(struct sigcontext));
526 if (dst_phys == 0) return(EFAULT);
527 phys_copy(vir2phys(&sc), dst_phys, (phys_bytes) sizeof(struct sigcontext));
528
529 /* Initialize the sigframe structure. */
530 frp = (struct sigframe *) scp - 1;
531 fr.sf_scpcopy = scp;
532 fr.sf_retadr2 = (void (*)()) rp->p_reg.pc;
533 fr.sf_fp = rp->p_reg.fp;
534 rp->p_reg.fp = (reg_t) &frp->sf_fp;
535 fr.sf_scp = scp;
536 fr.sf_code = 0; /* XXX - should be used for type of FP exception */
537 fr.sf_signo = smsg.sm_signo;
538 fr.sf_retadr = (void (*)()) smsg.sm_sigreturn;
539
540 /* Copy the sigframe structure to the user's stack. */
541 dst_phys = umap_local(rp, D, (vir_bytes) frp,
542 (vir_bytes) sizeof(struct sigframe));
543 if (dst_phys == 0) return(EFAULT);

```

Nov 05, 14 14:51

signal_related.txt

Page 10/14

```

544 phys_copy(vir2phys(&fr), dst_phys, (phys_bytes) sizeof(struct sigframe));
545
546 /* Reset user registers to execute the signal handler. */
547 rp->p_reg.sp = (reg_t) frp;
548 rp->p_reg.pc = (reg_t) smsg.sm_sighandler;
549
550 return(OK);
551 }
552
553 #endif /* USE_SIGSEND */
554

```

Nov 05, 14 14:51

signal_related.txt

Page 11/14

```

555 ***** kernel/system/do_sigreturn.c *****
556 /* The kernel call that is implemented in this file:
557 *   m_type:   SYS_SIGRETURN
558 *
559 * The parameters for this kernel call are:
560 *   m2_i1:   SIG_PROC      # process returning from handler
561 *   m2_p1:   SIG_CTXT_PTR  # pointer to sigcontext structure
562 *
563 */
564
565 #include "../system.h"
566 #include <string.h>
567 #include <signal.h>
568 #include <sys/sigcontext.h>
569
570 #if USE_SIGRETURN
571
572 /*=====
573 *                               do_sigreturn                               *
574 *=====*/
575 PUBLIC int do_sigreturn(m_ptr)
576 message *m_ptr;          /* pointer to request message */
577 {
578     /* POSIX style signals require sys_sigreturn to put things in order before
579     * the signalled process can resume execution
580     */
581     struct sigcontext sc;
582     register struct proc *rp;
583     phys_bytes src_phys;
584
585     if (!isokprocn(m_ptr->SIG_PROC)) return(EINVAL);
586     if (iskerneln(m_ptr->SIG_PROC)) return(EPERM);
587     rp = proc_addr(m_ptr->SIG_PROC);
588
589     /* Copy in the sigcontext structure. */
590     src_phys = umap_local(rp, D, (vir_bytes) m_ptr->SIG_CTXT_PTR,
591         (vir_bytes) sizeof(struct sigcontext));
592     if (src_phys == 0) return(EFAULT);
593     phys_copy(src_phys, vir2phys(&sc), (phys_bytes) sizeof(struct sigcontext));
594
595     /* Make sure that this is not just a jump buffer. */
596     if ((sc.sc_flags & SC_SIGCONTEXT) == 0) return(EINVAL);
597
598     /* Fix up only certain key registers if the compiler doesn't use
599     * register variables within functions containing setjmp.
600     */
601     if (sc.sc_flags & SC_NOREGLOCALS) {
602         rp->p_reg.retreg = sc.sc_retreg;
603         rp->p_reg.fp = sc.sc_fp;
604         rp->p_reg.pc = sc.sc_pc;
605         rp->p_reg.sp = sc.sc_sp;
606         return(OK);
607     }
608     sc.sc_psw = rp->p_reg.psw;
609
610     #if (CHIP == INTEL)
611         /* Don't panic kernel if user gave bad selectors. */
612         sc.sc_cs = rp->p_reg.cs;
613         sc.sc_ds = rp->p_reg.ds;
614         sc.sc_es = rp->p_reg.es;
615         #if _WORD_SIZE == 4
616             sc.sc_fs = rp->p_reg.fs;
617             sc.sc_gs = rp->p_reg.gs;
618         #endif
619         #endif
620
621         /* Restore the registers. */
622         memcpy(&rp->p_reg, &sc.sc_regs, sizeof(struct sigregs));
623         return(OK);
624     }
625 #endif /* USE_SIGRETURN */
626

```

Nov 05, 14 14:51

signal_related.txt

Page 12/14

```

627 ===== kernel/system/do_kill.c =====
628 /* The kernel call that is implemented in this file:
629 *   m_type:   SYS_KILL
630 *
631 * The parameters for this kernel call are:
632 *   m2_i1:   SIG_PROC      # process to signal/ pending
633 *   m2_i2:   SIG_NUMBER    # signal number to send to process
634 *
635 */
636 #include "../system.h"
637 #include <signal.h>
638 #include <sys/sigcontext.h>
639
640 #if USE_KILL
641
642 /*=====
643 *                               do_kill                               *
644 *=====*/
645 PUBLIC int do_kill(m_ptr)
646 message *m_ptr;          /* pointer to request message */
647 {
648     /* Handle sys_kill(). Cause a signal to be sent to a process. The PM is the
649     * central server where all signals are processed and handler policies can
650     * be registered. Any request, except for PM requests, is added to the map
651     * of pending signals and the PM is informed about the new signal.
652     * Since system servers cannot use normal POSIX signal handlers (because they
653     * are usually blocked on a RECEIVE), they can request the PM to transform
654     * signals into messages. This is done by the PM with a call to sys_kill().
655     */
656     proc_nr_t proc_nr = m_ptr->SIG_PROC;
657     int sig_nr = m_ptr->SIG_NUMBER;
658
659     if (!isokprocn(proc_nr) || sig_nr > _NSIG) return(EINVAL);
660     if (iskerneln(proc_nr)) return(EPERM);
661
662     if (m_ptr->m_source == PM_PROC_NR) {
663         /* Directly send signal notification to a system process. */
664         if (! (priv(proc_addr(proc_nr))->s_flags & SYS_PROC)) return(EPERM);
665         send_sig(proc_nr, sig_nr);
666     } else {
667         /* Set pending signal to be processed by the PM. */
668         cause_sig(proc_nr, sig_nr);
669     }
670     return(OK);
671 }
672
673 #endif /* USE_KILL */
674

```

Nov 05, 14 14:51

signal_related.txt

Page 13/14

```

675 ===== kernel/system/do_getksig.c =====
676 /* The kernel call that is implemented in this file:
677 *   m_type:   SYS_GETKSIG
678 *
679 * The parameters for this kernel call are:
680 *   m2_il:    SIG_PROC      # process with pending signals
681 *   m2_ll:    SIG_MAP      # bit map with pending signals
682 */
683
684 #include "../system.h"
685 #include <signal.h>
686 #include <sys/sigcontext.h>
687
688 #if USE_GETKSIG
689
690 /*=====
691 *                               do_getksig                               *
692 *=====*/
693 PUBLIC int do_getksig(m_ptr)
694 message *m_ptr;          /* pointer to request message */
695 {
696 /* PM is ready to accept signals and repeatedly does a kernel call to get
697 * one. Find a process with pending signals. If no signals are available,
698 * return NONE in the process number field.
699 * It is not sufficient to ready the process when PM is informed, because
700 * PM can block waiting for FS to do a core dump.
701 */
702 register struct proc *rp;
703
704 /* Find the next process with pending signals. */
705 for (rp = BEG_USER_ADDR; rp < END_PROC_ADDR; rp++) {
706     if (rp->p_rts_flags & SIGNED) {
707         m_ptr->SIG_PROC = rp->p_nr;          /* store signaled process */
708         m_ptr->SIG_MAP = rp->p_pending;      /* pending signals map */
709         sigemptyset(&rp->p_pending);        /* ball is in PM's court */
710         rp->p_rts_flags &= ~SIGNED;         /* blocked by SIG_PENDING */
711         return(OK);
712     }
713 }
714
715 /* No process with pending signals was found. */
716 m_ptr->SIG_PROC = NONE;
717 return(OK);
718 }
719 #endif /* USE_GETKSIG */
720

```

Nov 05, 14 14:51

signal_related.txt

Page 14/14

```

721 ===== kernel/system/do_endksig.c =====
722 /* The kernel call that is implemented in this file:
723 *   m_type:   SYS_ENDKSIG
724 *
725 * The parameters for this kernel call are:
726 *   m2_il:    SIG_PROC      # process for which PM is done
727 */
728
729 #include "../system.h"
730 #include <signal.h>
731 #include <sys/sigcontext.h>
732
733 #if USE_ENDKSIG
734
735 /*=====
736 *                               do_endksig                               *
737 *=====*/
738 PUBLIC int do_endksig(m_ptr)
739 message *m_ptr;          /* pointer to request message */
740 {
741 /* Finish up after a kernel type signal, caused by a SYS_KILL message or a
742 * call to cause_sig by a task. This is called by the PM after processing a
743 * signal it got with SYS_GETKSIG.
744 */
745 register struct proc *rp;
746
747 /* Get process pointer and verify that it had signals pending. If the
748 * process is already dead its flags will be reset.
749 */
750 rp = proc_addr(m_ptr->SIG_PROC);
751 if (! (rp->p_rts_flags & SIG_PENDING)) return(EINVAL);
752
753 /* PM has finished one kernel signal. Perhaps process is ready now? */
754 if (! (rp->p_rts_flags & SIGNED)) /* new signal arrived */
755     if ((rp->p_rts_flags & ~SIG_PENDING)==0) /* remove pending flag */
756         lock_enqueue(rp);                /* ready if no flags */
757     return(OK);
758 }
759
760 #endif /* USE_ENDKSIG */
761
762

```