```
/*        @(#)ieeefp.h 1.5 88/02/08 SMI      */

/*
 * Copyright (c) 1987 by Sun Microsystems, Inc.
 */

/*
                    #include <sys/ieeefp.h>

contains definitions for constants and types for IEEE floating point.

Source at     /usr/src/sys/sys/ieeefp.h
Install at    /usr/include/sys/ieeefp.h
*/

/*        Sun TYPES for IEEE floating point.          */

#ifdef sparc
enum fp_direction_type              /* rounding direction */
        {
        fp_nearest      = 0,
        fp_tozero       = 1,
        fp_positive     = 2,
        fp_negative     = 3
        } ;
#endif
#ifdef i386
enum fp_direction_type              /* rounding direction */
        {
        fp_nearest      = 0,
        fp_negative     = 1,
        fp_positive     = 2,
        fp_tozero       = 3
        } ;
#endif
#ifdef mc68000
enum fp_direction_type              /* rounding direction */
        {
        fp_nearest      = 0,
        fp_tozero       = 1,
        fp_negative     = 2,
        fp_positive     = 3
        } ;
#endif

#ifdef i386
enum fp_precision_type              /* extended rounding precision */
        {
        fp_single       = 0,
        fp_precision_3  = 1,
        fp_double       = 2,
        fp_extended     = 3
        } ;
#else
```

```
enum fp_precision_type              /* extended rounding precision */
        {
        fp_extended     = 0,
        fp_single       = 1,
        fp_double       = 2,
        fp_precision_3  = 3
        } ;
#endif

#ifdef i386
enum fp_exception_type              /* exceptions according to bit number */
        {
        fp_invalid      = 0,
        fp_denormalized = 1,
        fp_division     = 2,
        fp_overflow     = 3,
        fp_underflow    = 4,
        fp_inexact      = 5
        } ;
#else
enum fp_exception_type              /* exceptions according to bit number */
        {
        fp_inexact      = 0,
        fp_division     = 1,
        fp_underflow    = 2,
        fp_overflow     = 3,
        fp_invalid      = 4
        } ;
#endif

enum fp_class_type                  /* floating-point classes */
        {
        fp_zero         = 0,
        fp_subnormal    = 1,
        fp_normal       = 2,
        fp_infinity     = 3,
        fp_quiet = 4,
        fp_signaling    = 5
        } ;
```

```
/*      @(#)floatingpoint.h 1.8 88/02/07 SMI       */

/*
 * Copyright (c) 1987 by Sun Microsystems, Inc.
 */

/*
                    #include <floatingpoint.h>

contains definitions for constants, types, variables, and functions
implemented in libc.a for:
*       IEEE floating-point arithmetic base conversion;
*       IEEE floating-point arithmetic modes;
*       IEEE floating-point arithmetic exception handling;
*       certain functions defined in 4.3 BSD and System V.

Source at       /usr/src/include/floatingpoint.h
Install at      /usr/include/floatingpoint.h
*/

#include <sys/ieeefp.h>

/*      Sun TYPES for IEEE floating point.         */

typedef float single ;
typedef unsigned extended[3] ;

#define N_IEEE_EXCEPTION 5   /* Number of floating-point exceptions. */

typedef unsigned fp_exception_field_type ;
                                /*
                                A field containing fp_exceptions OR'ed
                                together.
                                 */

typedef int sigfpe_code_type ; /* Type of SIGFPE code. */

typedef void (* sigfpe_handler_type)() ;
                                /* Pointer to exception handler function. */

#define SIGFPE_DEFAULT (void (*)())0   /* default exception handling */
#define SIGFPE_IGNORE  (void (*)())1   /* ignore this exception or code */
#define SIGFPE_ABORT   (void (*)())2   /* force abort on exception */

/*      Sun VARIABLES for IEEE floating point. */

extern enum fp_direction_type fp_direction ;
                                /*
                                Current rounding direction.
                                Updated by ieee_flags.
                                */

extern enum fp_precision_type fp_precision ;
                                /*
```

```
                                        Current rounding precision.
                                        Updated by ieee_flags.
                                        */

extern sigfpe_handler_type ieee_handlers [N_IEEE_EXCEPTION] ;
                                        /*
                                        Array of pointers to functions
                                        to handle SIGFPE's corresponding
                                        to IEEE fp_exceptions.
                                        sigfpe_default means do not generate
                                        SIGFPE.
                                        An invalid address such as sigfpe_abort
                                        will cause abort on that SIGFPE.
                                        Updated by ieee_handler.
                                        */
extern fp_exception_field_type fp_accrued_exceptions ;
                                        /*
                                        Sticky accumulated exceptions, updated by
                                        ieee_flags.
                                        In hardware implementations this variable
                                        is not automatically updated as the hardware
                                        changes and should therefore not be relied
                                        on directly.
                                        */

extern sigfpe_handler_type sigfpe( ) ;

/*      Sun definitions for base conversion.                    */

#define DECIMAL_STRING_LENGTH 512
                                        /* Size of buffer in decimal_record. */

typedef char decimal_string[DECIMAL_STRING_LENGTH] ;
                                        /* Decimal significand. */

typedef struct
        {
        enum fp_class_type fpclass ;
        int     sign ;
        int     exponent ;
        decimal_string ds ;             /* Significand - each char contains an ascii
                                          digit, except the string-terminating
                                          ascii null. */
        int     more ;                  /* On conversion from decimal to binary, != 0
                                          indicates more non-zero digits following
                                          ds. */
        int     ndigits ; /* On fixed_form conversion from binary to
                                          decimal, contains number of digits required
                                          for ds. */
        }
        decimal_record ;

enum decimal_form
        {
```

```
        fixed_form,               /* Fortran F format: ndigits specifies number of
                                    digits after point; if negative, specifies
                                    rounding to occur to left of point. */
        floating_form             /* Fortran E format: ndigits specifies number of
                                    significant digits. */
        } ;

typedef struct
        {
        enum fp_direction_type rd ;
                                  /* Rounding direction. */
        enum decimal_form df ;    /* Format for conversion from
                                    binary to decimal. */
        int ndigits ;             /* Number of digits for conversion. */
        }
        decimal_mode ;

enum decimal_string_form
        {                         /* Valid decimal number string formats. */
        invalid_form,             /* Not a valid decimal string format. */
        whitespace_form,/* All white space - valid in Fortran! */
        fixed_int_form,           /* <digs>                    */
        fixed_intdot_form,        /* <digs>.                   */
        fixed_dotfrac_form,       /* .<digs>                   */
        fixed_intdotfrac_form,    /* <digs>.<frac> */
        floating_int_form,        /* <digs><exp>               */
        floating_intdot_form,     /* <digs>.<exp>              */
        floating_dotfrac_form,    /* .<digs><exp>              */
        floating_intdotfrac_form,/* <digs>.<digs><exp>         */
        inf_form,                 /* inf                       */
        infinity_form,            /* infinity                  */
        nan_form,                 /* nan                       */
        nanstring_form            /* nan(string)               */
        } ;

extern void single_to_decimal ( ) ;
extern void double_to_decimal ( ) ;
extern void extended_to_decimal ( ) ;

extern void decimal_to_single ( ) ;
extern void decimal_to_double ( ) ;
extern void decimal_to_extended ( ) ;

extern char *econvert( ) ;
extern char *fconvert( ) ;
extern char *gconvert( ) ;
extern char *seconvert( ) ;
extern char *sfconvert( ) ;
extern char *sgconvert( ) ;

extern void string_to_decimal( ) ;
extern void  file_to_decimal( ) ;
extern void  func_to_decimal( ) ;
```

```
/*          Definitions from 4.3 BSD math.h    4.6   9/11/85         */

extern double atof();

/*          Definitions from System V                                */

extern int errno;

extern double strtod ();
```

```
/*       @(#)math.h 1.23 88/03/03 SMI      */

/*
 * Copyright (c) 1988 by Sun Microsystems, Inc.
 */

/*
                        #include <math.h>

defines all the public functions implemented in libm.a.

*/

#ifndef M_SQRT1_2
#include <floatingpoint.h>          /* Contains definitions for types and
                                       functions implemented in libc.a.  */

/*       4.3 BSD functions: math.h4.6       9/11/85  */

extern int    finite();
extern double fabs(), floor(), ceil(), rint();
extern double hypot();
extern double copysign();
extern double sqrt();
extern double modf(), frexp();
extern double asinh(), acosh(), atanh();
extern double erf(), erfc();
extern double exp(), expm1(), log(), log10(), log1p(), pow();
extern double lgamma();
extern double j0(), j1(), jn(), y0(), y1(), yn();
extern double sin(), cos(), tan(), asin(), acos(), atan(), atan2();
extern double sinh(), cosh(), tanh();
extern double cbrt();

/*     Sun definitions.      */

enum fp_pi_type {                   /* Implemented precisions for trigonometric
                                       argument reduction. */
        fp_pi_infinite    = 0,      /* Infinite-precision approximation to pi. */
        fp_pi_66          = 1,      /* 66-bit approximation to pi. */
        fp_pi_53          = 2       /* 53-bit approximation to pi. */
        } ;

extern enum fp_pi_type fp_pi ; /* Pi precision to use for trigonometric
                                       argument reduction. */

/*       Functions callable from C, intended to support IEEE arithmetic.       */

extern enum fp_class_type fp_class() ;
extern int ilogb(), irint(), signbit() ;
extern int isinf(), isnan(), isnormal(), issubnormal(), iszero() ;
extern double nextafter(), remainder() ;
extern double logb(), significand(), scalb(), scalbn();
extern double min_subnormal(), max_subnormal();
extern double min_normal(), max_normal();
extern double infinity(), quiet_nan(), signaling_nan();

/*	Functions callable from C, intended to support Fortran.		*/

extern double log2(), exp10(), exp2(), aint(), anint() ;
extern int nint() ;
extern void   sincos();

/*	Sun FUNCTIONS for C Programmers for IEEE floating point. */

extern int ieee_flags ();
extern int ieee_handler ();

/*	Single-precision functions callable from Fortran, Pascal, Modula-2, etc.,
	take float* arguments instead of double and
	return FLOATFUNCTIONTYPE results instead of double.
	RETURNFLOAT is used to return a float function value without conversion to
	double.
	ASSIGNFLOAT is used to get the float value out of a FLOATFUNCTIONTYPE result.
	We don't want you to have to think about -fsingle2.   *

	Some internal library functions pass float parameters as 32-bit values,
	disguised as FLOATPARAMETER.  FLOATPARAMETERVALUE(x) extracts the
	float value from the FLOATPARAMETER.
*/

/*	mc68000 returns float results in d0, same as int	*/

#ifdef mc68000
#define FLOATFUNCTIONTYPE  int
#define RETURNFLOAT(x)			return (*(int *)(&(x)))
#define ASSIGNFLOAT(x,y)	*(int *)(&x) = y
#endif

/*	sparc returns float results in %f0, same as top half of double	*/

#ifdef sparc
#define FLOATFUNCTIONTYPE  double
#define RETURNFLOAT(x)			{ union {double _d ; float _f } _kluge ; _kluge._f = (x) ; return _kluge._d ; }
#define ASSIGNFLOAT(x,y)	{ union {double _d ; float _f } _kluge ; _kluge._d = (y) ; x = _kluge._f ; }
#endif

/*	i386 returns float results on stack as extendeds, same as double */

#ifdef i386
#define FLOATFUNCTIONTYPE  float
#define RETURNFLOAT(x)			return (x)
#define ASSIGNFLOAT(x,y)	x = y
#endif

/*	So far everybody passes float parameters as 32 bits on stack, same as int.	*/

#define FLOATPARAMETER			int
```

```
#define FLOATPARAMETERVALUE(x)    (*(float *)(&(x)))

extern int    ir_finite_();
extern FLOATFUNCTIONTYPE r_fabs_(), r_floor_(), r_ceil_(), r_rint_();
extern FLOATFUNCTIONTYPE r_hypot_();
extern FLOATFUNCTIONTYPE r_copysign_();
extern FLOATFUNCTIONTYPE r_sqrt_();
extern FLOATFUNCTIONTYPE r_asinh_(), r_acosh_(), r_atanh_();
extern FLOATFUNCTIONTYPE r_erf_(), r_erfc_();
extern FLOATFUNCTIONTYPE r_exp_(), r_expm1_(), r_log_(), r_log10_(), r_log1p_();
extern FLOATFUNCTIONTYPE r_pow_();
extern FLOATFUNCTIONTYPE r_lgamma_();
extern FLOATFUNCTIONTYPE r_j0_(), r_j1_(), r_jn_(), r_y0_(), r_y1_(), r_yn_();
extern FLOATFUNCTIONTYPE r_sin_(), r_cos_(), r_tan_(), r_asin_(), r_acos_();
extern FLOATFUNCTIONTYPE r_atan_(), r_atan2_();
extern FLOATFUNCTIONTYPE r_sinh_(), r_cosh_(), r_tanh_();
extern FLOATFUNCTIONTYPE r_cbrt_();
extern int ir_ilogb_(), ir_irint_(), ir_signbit_() ;
extern int ir_isinf_(), ir_isnan_(),
            ir_issubnormal_(), ir_isnormal_(), ir_iszero_() ;
extern enum fp_class_type ir_fp_class_();
extern FLOATFUNCTIONTYPE r_nextafter_(), r_remainder_() ;
extern FLOATFUNCTIONTYPE r_log2_(), r_exp10_(), r_exp2_(), r_aint_(), r_anint_() ;
extern int ir_nint_() ;
extern FLOATFUNCTIONTYPE r_fmod_();
extern FLOATFUNCTIONTYPE r_logb_(), r_significand_(), r_scalb_(), r_scalbn_();
extern FLOATFUNCTIONTYPE r_min_subnormal_(), r_max_subnormal_();
extern FLOATFUNCTIONTYPE r_min_normal_(), r_max_normal_();
extern FLOATFUNCTIONTYPE r_infinity_(), r_quiet_nan_(), r_signaling_nan_();
extern void r_sincos_();

/*        Constants, variables, and functions from System V */

#define _ABS(x) ((x) < 0 ? -(x) : (x))

#define  HUGE_VAL      (infinity())        /* Produces IEEE Infinity. */
#define  HUGE          (infinity())        /* For historical compatibility. */

#define DOMAIN       1
#define SING         2
#define OVERFLOW     3
#define UNDERFLOW    4
#define TLOSS        5
#define PLOSS        6

struct exception {
    int type;
    char *name;
    double arg1;
    double arg2;
    double retval;
};

extern int signgam;
extern double fmod();
extern int matherr();

/* First three have to be defined exactly as in values.h including spacing! */

#define M_LN2  0.69314718055994530942
#define M_PI     3.14159265358979323846
#define M_SQRT2         1.41421356237309504880

#define M_E             2.7182818284590452354
#define M_LOG2E                   1.4426950408889634074
#define M_LOG10E        0.43429448190325182765
#define M_LN10          2.30258509299404568402
#define M_PI_2          1.57079632679489661923
#define M_PI_4          0.78539816339744830962
#define M_1_PI          0.31830988618379067154
#define M_2_PI          0.63661977236758134308
#define M_2_SQRTPI      1.12837916709551257390
#define M_SQRT1_2       0.70710678118654752440
#define _POLY1(x, c)  ((c)[0] * (x) + (c)[1])
#define _POLY2(x, c)  (_POLY1((x), (c)) * (x) + (c)[2])
#define _POLY3(x, c)  (_POLY2((x), (c)) * (x) + (c)[3])
#define _POLY4(x, c)  (_POLY3((x), (c)) * (x) + (c)[4])
#define _POLY5(x, c)  (_POLY4((x), (c)) * (x) + (c)[5])
#define _POLY6(x, c)  (_POLY5((x), (c)) * (x) + (c)[6])
#define _POLY7(x, c)  (_POLY6((x), (c)) * (x) + (c)[7])
#define _POLY8(x, c)  (_POLY7((x), (c)) * (x) + (c)[8])
#define _POLY9(x, c)  (_POLY8((x), (c)) * (x) + (c)[9])

/*
        Deprecated functions for compatibility with past.
        Changes planned for future.
*/

extern double cabs();      /* Use double hypot(x,y)
                              Traditional cabs usage is confused -
                              is its argument two doubles or one struct?  */
extern double drem();      /* Use double remainder(x,y)
                              drem will disappear in a future release.    */
extern double gamma();     /* Use double lgamma(x)
                              to compute log of gamma function.
                              Name gamma is reserved for true gamma function
                              to appear in a future release.              */
extern double ldexp();     /* Use double scalbn(x,n)
                              ldexp may disappear in a future release     */

#endif
```

```
/*	@(#)signal.h 2.29 88/03/03 SMI; from UCB 6.7 85/06/08	*/

/*
 * Copyright (c) 1982 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

/*
 * Copyright (c) 1987 by Sun Microsystems, Inc.
 */

#ifndef _sys_signal_h
#define _sys_signal_h
#include <vm/faultcode.h>
#define NSIG	32

#define	SIGHUP	1	/* hangup */
#define	SIGINT 2	/* interrupt */
#define	SIGQUIT	3	/* quit */
#define	SIGILL 4	/* illegal instruction (not reset when caught) */
#ifdef vax
#define	ILL_RESAD_FAULT	0x0	/* reserved addressing fault */
#define	ILL_PRIVIN_FAULT	0x1	/* privileged instruction fault */
#define	ILL_RESOP_FAULT	0x2	/* reserved operand fault */
/* CHME, CHMS, CHMU are not yet given back to users reasonably */
#endif vax
#ifdef mc68000
#define	ILL_ILLINSTR_FAULT	0x10	/* illegal instruction fault */
#define	ILL_PRIVVIO_FAULT 0x20	/* privilege violation fault */
#define	ILL_COPROCERR_FAULT	0x34	/* [coprocessor protocol error fault] */
#define	ILL_TRAP1_FAULT	0x84	/* trap #1 fault */
#define	ILL_TRAP2_FAULT	0x88	/* trap #2 fault */
#define	ILL_TRAP3_FAULT	0x8c	/* trap #3 fault */
#define	ILL_TRAP4_FAULT	0x90	/* trap #4 fault */
#define	ILL_TRAP5_FAULT	0x94	/* trap #5 fault */
#define	ILL_TRAP6_FAULT	0x98	/* trap #6 fault */
#define	ILL_TRAP7_FAULT	0x9c	/* trap #7 fault */
#define	ILL_TRAP8_FAULT	0xa0	/* trap #8 fault */
#define	ILL_TRAP9_FAULT	0xa4	/* trap #9 fault */
#define	ILL_TRAP10_FAULT	0xa8	/* trap #10 fault */
#define	ILL_TRAP11_FAULT	0xac	/* trap #11 fault */
#define	ILL_TRAP12_FAULT	0xb0	/* trap #12 fault */
#define	ILL_TRAP13_FAULT	0xb4	/* trap #13 fault */
#define	ILL_TRAP14_FAULT	0xb8	/* trap #14 fault */
#endif mc68000
#ifdef sparc
#define	ILL_STACK	0x00	/* bad stack */
#define	ILL_ILLINSTR_FAULT	0x02	/* illegal instruction fault */
#define	ILL_PRIVINSTR_FAULT	0x03	/* privileged instruction fault */
/* codes from 0x80 to 0xff are software traps */
#define	ILL_TRAP_FAULT(n)	((n)+0x80) /* trap n fault */
#endif sparc
#define	SIGTRAP	5	/* trace trap (not reset when caught) */
```

```
#define  SIGIOT 6        /* IOT instruction */
#define SIGABRT 6        /* used by abort, replace SIGIOT in the future */
#define  SIGEMT          7          /* EMT instruction */
#ifdef mc68000
#define   EMT_EMU1010                    0x28     /* line 1010 emulator trap */
#define   EMT_EMU1111                    0x2c     /* line 1111 emulator trap */
#endif mc68000
#ifdef sparc
#define    EMT_TAG               0x0a     /* tag overflow */
#endif sparc
#define  SIGFPE 8        /* floating point exception */
#ifdef vax
#define   FPE_INTOVF_TRAP   0x1      /* integer overflow */
#define   FPE_INTDIV_TRAP   0x2      /* integer divide by zero */
#define   FPE_FLTOVF_TRAP   0x3      /* floating overflow */
#define   FPE_FLTDIV_TRAP   0x4      /* floating/decimal divide by zero */
#define   FPE_FLTUND_TRAP   0x5      /* floating underflow */
#define   FPE_DECOVF_TRAP   0x6      /* decimal overflow */
#define   FPE_SUBRNG_TRAP   0x7      /* subscript out of range */
#define   FPE_FLTOVF_FAULT  0x8      /* floating overflow fault */
#define   FPE_FLTDIV_FAULT  0x9      /* divide by zero floating fault */
#define   FPE_FLTUND_FAULT  0xa      /* floating underflow fault */
#endif vax
#ifdef mc68000
#define   FPE_INTDIV_TRAP   0x14     /* integer divide by zero */
#define   FPE_CHKINST_TRAP  0x18     /* CHK [CHK2] instruction */
#define   FPE_TRAPV_TRAP    0x1c     /* TRAPV [cpTRAPcc TRAPcc] instr */
#define   FPE_FLTBSUN_TRAP  0xc0     /* [branch or set on unordered cond] */
#define   FPE_FLTINEX_TRAP  0xc4     /* [floating inexact result] */
#define   FPE_FLTDIV_TRAP   0xc8     /* [floating divide by zero] */
#define   FPE_FLTUND_TRAP   0xcc     /* [floating underflow] */
#define   FPE_FLTOPERR_TRAP          0xd0     /* [floating operand error] */
#define   FPE_FLTOVF_TRAP   0xd4     /* [floating overflow] */
#define   FPE_FLTNAN_TRAP   0xd8     /* [floating Not-A-Number] */
#ifdef sun
#define   FPE_FPA_ENABLE    0x400    /* [FPA not enabled] */
#define   FPE_FPA_ERROR     0x404    /* [FPA arithmetic exception] */
#endif sun
#endif mc68000
#ifdef sparc
#define   FPE_INTOVF_TRAP   0x1      /* integer overflow */
#define   FPE_INTDIV_TRAP   0x14     /* integer divide by zero */
#define   FPE_FLTINEX_TRAP  0xc4     /* [floating inexact result] */
#define   FPE_FLTDIV_TRAP   0xc8     /* [floating divide by zero] */
#define   FPE_FLTUND_TRAP   0xcc     /* [floating underflow] */
#define   FPE_FLTOPERR_TRAP          0xd0     /* [floating operand error] */
#define   FPE_FLTOVF_TRAP   0xd4     /* [floating overflow] */
#endif sparc
#define  SIGKILL         9          /* kill (cannot be caught or ignored) */
/*
 * The codes for SIGBUS and SIGSEGV are described in <vm/faultcode.h>
 */
#define  SIGBUS10        /* bus error */
#define   BUS_HWERR  FC_HWERR       /* misc hardware error (e.g. timeout) */
```

```
#define   BUS_ALIGN   FC_ALIGN       /* hardware alignment error */
#define  SIGSEGV        11       /* segmentation violation */
#define   SEGV_NOMAP         FC_NOMAP      /* no mapping at the fault address */
#define   SEGV_PROT  FC_PROT              /* access exceeded protections */
#define   SEGV_OBJERR        FC_OBJERR     /* object returned errno value */
/*
 * The SEGV_CODE(code) will be SEGV_NOMAP, SEGV_PROT, or SEGV_OBJERR.
 * In the SEGV_OBJERR case, doing a SEGV_ERRNO(code) gives an errno value
 * reported by the underlying file object mapped at the fault address.
 */
#define   SEGV_CODE(C)        FC_CODE(C)
#define   SEGV_ERRNO(C)       FC_ERRNO(C)
#define  SIGSYS 12       /* bad argument to system call */
#define  SIGPIPE         13        /* write on a pipe with no one to read it */
#define  SIGALRM         14        /* alarm clock */
#define  SIGTERM         15        /* software termination signal from kill */
#define  SIGURG          16        /* urgent condition on IO channel */
#define  SIGSTOP         17        /* sendable stop signal not from tty */
#define  SIGTSTP         18        /* stop signal from tty */
#define  SIGCONT         19        /* continue a stopped process */
#define  SIGCHLD         20        /* to parent on child stop or exit */
#define  SIGCLD          20        /* System V name for SIGCHLD */
#define  SIGTTIN         21        /* to readers pgrp upon background tty read */
#define  SIGTTOU         22        /* like TTIN for output if (tp->t_local&LTOSTOP) */
#define  SIGIO   23      /* input/output possible signal */
#define  SIGPOLL         SIGIO   /* System V name for SIGIO */
#define  SIGXCPU         24        /* exceeded CPU time limit */
#define  SIGXFSZ         25        /* exceeded file size limit */
#define  SIGVTALRM 26/* virtual time alarm */
#define  SIGPROF         27        /* profiling time alarm */
#define  SIGWINCH 28     /* window changed */
#define  SIGLOST 29      /* resource lost (eg, record-lock lost) */
#define SIGUSR1 30       /* user defined signal 1 */
#define SIGUSR2 31       /* user defined signal 2 */
/*
 * If addr cannot be computed it is set to SIG_NOADDR.
 */
#define SIG_NOADDR    ((char *)~0)

#ifndef KERNEL
void     (*signal())();
/*
 * Define BSD 4.1 reliable signals for SVID compatibility.
 * These functions may go away in a future release.
 */
void  (*sigset())();
int  sighold();
int  sigrelse();
int  sigignore();
#endif !KERNEL

#ifndef LOCORE
/*
 * Signal vector "template" used in sigvec call.
 */
struct   sigvec {
         void     (*sv_handler)();  /* signal handler */
         int      sv_mask;                 /* signal mask to apply */
         int      sv_flags;         /* see signal options below */
};
#define SV_ONSTACK    0x0001  /* take signal on signal stack */
#define SV_INTERRUPT 0x0002  /* do not restart system on signal return */
#define SV_RESETHAND          0x0004  /* reset signal handler to SIG_DFL when signal taken */
#define sv_onstack sv_flags           /* isn't compatibility wonderful! */

/*
 * Structure used in sigstack call.
 */
struct   sigstack {
         char     *ss_sp;                  /* signal stack pointer */
         int      ss_onstack;              /* current status */
};

/*
 * Information pushed on stack when a signal is delivered.
 * This is used by the kernel to restore state following
 * execution of the signal handler.  It is also made available
 * to the handler to allow it to properly restore state if
 * a non-standard exit is performed.
 */
struct   sigcontext {
         int      sc_onstack;              /* sigstack state to restore */
         int      sc_mask;                 /* signal mask to restore */
#ifdef vax
         int      sc_sp;                   /* sp to restore */
         int      sc_fp;                   /* fp to restore */
         int      sc_ap;                   /* ap to restore */
         int      sc_pc;                   /* pc to restore */
         int      sc_ps;                   /* psl to restore */
#endif vax
#ifdef mc68000
         int      sc_sp;                   /* sp to restore */
         int      sc_pc;                   /* pc to retore */
         int      sc_ps;                   /* psl to restore */
#endif mc68000
#ifdef sparc
#define MAXWINDOW   31                     /* max usable windows in sparc */
         int      sc_sp;                   /* sp to restore */
         int      sc_pc;                   /* pc to retore */
         int      sc_npc;                  /* next pc to restore */
         int      sc_psr;                  /* psr to restore */
         int      sc_g1;                   /* register that must be restored */
         int      sc_o0;
         int      sc_wbcnt;                /* number of outstanding windows */
         char     *sc_spbuf[MAXWINDOW];            /* sp's for each wbuf */
         int      sc_wbuf[MAXWINDOW][16];          /* outstanding window save buffer */
#endif sparc
#ifdef sun386
```

```
        int     sc_sp;                  /* sp to restore */
        int     sc_pc;                  /* pc to retore */
        int     sc_ps;                  /* psl to restore */
        int     sc_eax;                 /* eax to restore */
        int     sc_edx;                 /* edx to restore */
#endif
};
#endif !LOCORE

#define  BADSIG             (void (*)())-1
#define  SIG_ERR            (void (*)())-1
#define  SIG_DFL            (void (*)())0
#define  SIG_IGN            (void (*)())1

#ifdef KERNEL
#define  SIG_CATCH     (void (*)())2
#endif KERNEL
#define  SIG_HOLD      (void (*)())3

/*
 * Macro for converting signal number to a mask suitable for sigblock().
 */
#define sigmask(m)     (1 << ((m)-1))
#endif   !_sys_signal_h
```