Posix90

Fortran 95 Posix bindings

by Klaus Ramstock

1 Overview of Posix90

1.1 About Posix90

Why Posix90? After all, you who read this document know most likely enough about mixed-language programming so you could write your own stubs. But such things tend to be quick hacks, and I wanted a clean solution: Here it is!

1.2 Portability

At present, Posix90 is limited to gfortran and GNU/Linux. This might change if volunteers are found which port.

2 Linking against the Posix90 library

It is assumed here that you build the library from source. This is necessary, as the .mod files[1] which gfortran emits are still in a state of flux. It is too much work to provide them for all formats, compiling yourself is by far the best solution.

Place all .mod files in one directory, say /usr/local/include/f90 and place libposix90.a in another- say /usr/local/lib. You then need three compiler switches:

- -I /usr/local/include/f90 , to tell the compiler where to search for .mod files
- -lposix90, to tell the compiler to bind against libposix90.a
- -L/usr/local/lib , to tell the compiler where to find libposix90.a

Note that there is a space after -I, but no space after -L.

[1] .mod files hold the information which data and which routines constitute a module. They are created as a module is compiled and are needed when this module is used elsewhere

3 Using the library

3.1 Calling and naming conventions

Wherever possible, the names of the original posix routines are used. char * arguments are mapped to character(len=*)[1]. If the string is intent(out), an optional len argument is given to allow for trailing blanks. Int simply maps to integer. Small typedef'ed types like pid_t map to integer(pid_kind). More complex types like FILE map to type(FILE). Note that you should never pass derived types directly to C, as they are build alike their C pendants, but lack padding, makeing them assignment incompatible! Another issue here is the -fpackderived compiler switch.

All routines which possibly fail have an optional errno argument. If present, an error condition (0 == no error) is returned. If not present, an error message is printed if appropriate and execution stopped. All routines reset errno on entry. This behaviour is different from C, but makes it much easier to attribute an error to the offending routine.

[1] Be carefull with trailing spaces, use trim() or string(1:len) a lot

4 Posix90 modules available

4.1 Module f90_unix_dir

4.1.1 Parameters and Types

```
integer, parameter :: mode_kind
```

Integer kind used for file permissions.

4.1.2 Subroutines and Functions

```
subroutine chdir(path, errno)
character(len=*), intent(in) :: path
integer, intent(out), optional :: errno
```

See man 2 chdir

```
subroutine getcwd(path, lenpath, errno)
character(len=*), intent(out) :: path
integer, intent(out), optional :: lenpath, errno
```

See man 2 getcwd

```
subroutine link(existing, new, errno)
character(len=*), intent(in) :: existing, new
integer, intent(out), optional :: errno
```

See man 2 link

```
subroutine mkdir(path, mode, errno)
character(len=*), intent(in) :: path
integer(mode_kind), intent(in) :: mode
integer, intent(out), optional :: errno
```

See man 2 mkdir

```
subroutine mkfifo(path, mode, errno)
character(len=*), intent(in) :: path
integer(mode_kind), intent(in) :: mode
integer, intent(out), optional :: errno
```

See man 2 mkfifo

```
subroutine rmdir(path, errno)
character(len=*), intent(in) :: path
integer, intent(out), optional :: errno
```

See man 2 rmdir

```
subroutine unlink(path, errno)
character(len=*), intent(in) :: path
integer, intent(out), optional :: errno
```

See man 2 unlink

4.2 Module f90_unix_dirent

4.2.1 Parameters and Types

```
integer, parameter :: dir_kind
type DIR
   integer(dir_kind):: dir
end type DIR
```

Complex type handled by the routines in this section. You should not need to access its components directly.

4.2.2 Subroutines and Functions

```
subroutine closedir(dirp, errno)
  type(dir), intent(inout) :: dirp
  integer, intent(out), optional :: errno
```

See man 3 closedir.

```
subroutine opendir(dirname, dirp, errno)
  character(len=*), intent(in) :: dirname
  type(dir), intent(inout) :: dirp
  integer, intent(out), optional :: errno
```

See man 3 opendir.

```
subroutine readdir(dirp, name, lenname, errno)
  type(dir), intent(inout) :: dirp
  character(len=*), intent(out) :: name
  integer, intent(out) :: lenname
  integer, intent(out), optional :: errno
```

See man 3 readdir. The lenname argument is needed in case of trailing blanks in `name`.

```
subroutine rewinddir(dirp, errno)
  type(dir), intent(inout) :: dirp
  integer, intent(out), optional :: errno
```

See man 3 rewinddir.

4.3 Module f90_unix_env

4.3.1 Parameters and Types

```
integer, parameter :: CLOCK_KIND
integer, parameter :: LONG_KIND
integer, parameter :: GID_KIND
integer, parameter :: UID_KIND
integer, parameter :: PID_KIND
integer, parameter :: SIZET_KIND
```

Integer kind parameters for various data types.

```
integer, parameter :: NULL
```

```
integer, parameter :: L_CTERMID
```

Max. length of the result of `ctermid()`.

```
integer, parameter :: SC_ARG_MAX
integer, parameter :: SC_CHILD_MAX
integer, parameter :: SC_HOST_NAME_MAX
integer, parameter :: SC_LOGIN_NAME_MAX
integer, parameter :: SC_CLK_TCK
integer, parameter :: SC_OPEN_MAX
integer, parameter :: SC_PAGESIZE
integer, parameter :: SC_RE_DUP_MAX
integer, parameter :: SC_STREAM_MAX
integer, parameter :: SC_SYMLOOP_MAX
integer, parameter :: SC_TTY_NAME_MAX
integer, parameter :: SC_TZNAME_MAX
integer, parameter :: SC_VERSION
```

Possible values for the name argument to `sysconf()`.

```
type tms
   sequence
   integer(clock_kind):: utime, stime,cutime,cstime
```

```
end type tms
```

Structure used in `times()`. See man 2 times.

```
type utsname
   sequence
   character(len=80):: sysname, nodename, release, version, machine
end type utsname
```

Structure used in uname(). See man 2 uname.

4.3.2 Subroutines and Functions

```
integer(kind=clock_kind) function clk_tck()
```

Returns the clock ticks per second.

```
character(len=L_CTERMID) function ctermid(len)
  integer, intent(out), optional :: len
subroutine getarg2(k, arg, lenarg, errno)
  integer, intent(in) :: K
  character(len=*), intent(out), optional :: arg
  integer, intent(out), optional :: lenarg, errno
```

This routine is called `getarg2` to avoid a name clash with the build-in `getarg`. Note that `lenarg` is at present always `len_trim(arg)` .

```
integer(GID_KIND) function getegid()
subroutine getenv2(name, value, lenvalue, errno)
  character(len=*), intent(in) :: name
  character(len=*), intent(out), optional :: value
  integer, intent(out), optional :: lenvalue, errno
```

This routine is called `getenv2` to avoid a name clash with the build-in `getenv`. Note that `lenvalue` is at present always `len_trim(value)` .

```
integer(uid_kind) function geteuid()
  integer(uid_kind),external::c_geteuid
  geteuid = c_geteuid()
integer(gid_kind) function getgid()
subroutine getgroups(grouplist, ngroups, errno)
  integer(gid_kind), optional :: grouplist(:)
  integer, optional, intent(out) :: ngroups, errno
subroutine gethostname(name, lenname, errno)
  character(len=*), optional, intent(out) :: name
  integer, optional, intent(out) :: lenname, errno
subroutine getlogin(name, lenname, errno)
  character(len=*), optional, intent(out) :: name
  integer, optional, intent(out) :: lenname, errno
integer(PID_KIND) function getpgrp()
integer(PID_KIND) function getppid()
integer(UID_KIND) function getuid()
subroutine setgid(gid, errno)
    integer(GID_KIND), intent(in) :: gid
    integer, intent(out), optional :: errno
subroutine setpgid(gid, pgid, errno)
  integer(GID_KIND), intent(in) :: gid, pgid
  integer, intent(out), optional :: errno
subroutine setsid(errno)
  integer, intent(out), optional :: errno
```

```
subroutine setuid(gid, errno)
  integer(UID_KIND), intent(in) :: gid
  integer, intent(out), optional :: errno
subroutine sysconf(name, val,errno)
    integer, intent(in) :: name
    integer(long_kind), intent(out) :: val
    integer, intent(out), optional :: errno
integer(TIME_KIND) function time(errno)
  integer, optional, intent(out) :: errno
  integer(TIME_KIND), external :: c_time
integer(CLOCK_KIND) function times(buffer, errno)
    type(tms) :: buffer
    integer, optional, intent(out) :: errno
subroutine uname(name, errno)
  type(utsname), intent(out) :: name
  integer, optional, intent(out) :: errno
```

4.4 Module f90_unix_errno

4.4.1 Parameters and Types

```
integer, parameter :: E2BIG
integer, parameter :: EACCES
integer, parameter :: EAGAIN
integer, parameter :: EBADF
integer, parameter :: EBUSY
integer, parameter :: ECHILD
integer, parameter :: EDEADLK
integer, parameter :: EDOM
integer, parameter :: EEXIST
integer, parameter :: EFAULT
integer, parameter :: EFBIG
integer, parameter :: EINTR
integer, parameter :: EINVAL
integer, parameter :: EIO
integer, parameter :: EISDIR
integer, parameter :: EMFILE
integer, parameter :: EMLINK
integer, parameter :: ENAMETOOLONG
integer, parameter :: ENFILE
integer, parameter :: ENODEV
integer, parameter :: ENOENT
integer, parameter :: ENOEXEC
integer, parameter :: ENOLCK
integer, parameter :: ENOMEM
integer, parameter :: ENOSPC
integer, parameter :: ENOSYS
integer, parameter :: ENOTDIR
integer, parameter :: ENOTEMPTY
integer, parameter :: ENOTTY
integer, parameter :: ENXIO
integer, parameter :: EPERM
integer, parameter :: EPIPE
integer, parameter :: ERANGE
integer, parameter :: EROFS
integer, parameter :: ESPIPE
integer, parameter :: ESRCH
```

```
integer, parameter :: EXDEV
```

Error codes known to the module. Their names match the C equivalents.

4.4.2 Subroutines and Functions

```
character(len=80) function strerror(err, errno)
  integer, intent(in) :: err
  integer, intent(out), optional :: errno
```

See man 3 strerror.

```
subroutine perror(str, errc)
  character(len=*), intent(in) :: str
  integer, intent(in), optional :: errc
```

See man 3 perror.

```
integer function get_errno()
```

Get the value of errno.

```
subroutine set_errno(errc)
  integer, intent(in), optional :: errc
```

Set the value of errno to `errc` or 0, if `errc` is missing.

4.5 Module f90_unix_file

4.5.1 Parameters and Types

```
use f90_unix_tools, only : C0
use f90_unix_dir, only : mode_kind
use f90_unix_env, only : uid_kind, gid_kind
use f90_unix_time, only : time_kind
```

Constants used from other modules.

```
integer, parameter :: f_ok
integer, parameter :: r_ok
integer, parameter :: W_ok
integer, parameter :: X_ok
```

Constants used for access.

```
integer, parameter :: S_IRGRP
integer, parameter :: S_IROTH
integer, parameter :: S_IRUSR
integer, parameter :: S_IRWXG
integer, parameter :: S_IRWXO
integer, parameter :: S_IRWXU
integer, parameter :: S_ISGID
integer, parameter :: S_ISUID
integer, parameter :: S_IWGRP
integer, parameter :: S_IWOTH
integer, parameter :: S_IWUSR
integer, parameter :: S_IXGRP
integer, parameter :: S_IXOTH
integer, parameter :: S_IXUSR
```

Constants used for chmod.

```
integer, parameter :: dev_kind
integer, parameter :: ino_kind
integer, parameter :: off_kind
integer, parameter :: nlink_kind
```

Various kinds used in stat_t

```
type stat_t
   integer(dev_kind) ::     st_dev
   integer(ino_kind) ::     st_ino
   integer(mode_kind) ::    st_mode
   integer(nlink_kind) ::   st_nlink
   integer(uid_kind) ::     st_uid
   integer(gid_kind) ::     st_gid
   integer(dev_kind) ::     st_rdev
   integer(off_kind) ::     st_size
   integer(time_kind) ::    st_atime
   integer(time_kind) ::    st_mtime
   integer(time_kind) ::    st_ctime
end type stat_t
```

Type returned from stat. See man 2 stat for a field description.

```
type utimbuf
   integer(time_kind) :: actime, modtime
end type utimbuf
```

4.5.2 Subroutines and Functions

```
subroutine access(path, amode, errno)
  character(len=*), intent(in) :: path
  integer, intent(in) :: amode
  integer, intent(out) :: errno
```

See man 2 access. Use of this function can often be replaced by iostat().

```
subroutine chmod(path, mode, errno)
  character(len=*), intent(in) :: path
  integer(mode_kind), intent(in) :: mode
  integer, optional, intent(out) :: errno
```

See man 2 chmod.

```
subroutine chown(path, owner, group, errno)
  character(len=*), intent(in) :: path
  integer(UID_KIND), intent(in) :: owner
  integer(GID_KIND), intent(in) :: group
  integer, optional, intent(out) :: errno
```

See man 2 chown

```
subroutine stat(path, buf, errno)
  character(len=*), intent(in) :: path
  type(stat_t), intent(out) :: buf
  integer, optional, intent(out) :: errno
```

See man 2 stat.

4.6 Module f90_unix_io

4.6.1 Parameters and Types

```
use f90_unix_errno
use f90_unix_env, only : sizet_kind, NULL
use f90_unix_tools, only : C0
```

Constants used in this module.

```
integer, parameter :: EOF=-1
```

Definition of EOF

```
integer, parameter :: FILE_KIND
integer, parameter :: LONG_KIND
```

Kinds used in this module.

```
integer, parameter :: SEEK_SET
integer, parameter :: SEEK_CUR
integer, parameter :: SEEK_END
```

Constants used for fseek.

```
integer, parameter :: double
```

kind of 1.D0.

```
type FILE
   integer(file_kind) :: fp
end type FILE
```

Definition of FILE. There is no need to access fp directly.

```
interface fread
   module procedure fread_str, fread_str_array
   module procedure fread_real, fread_real_array
   module procedure fread_double, fread_double_array
   module procedure fread_int, fread_int_array
end interface
```

Interface for fread.

```
interface fwrite
   module procedure fwrite_str, fwrite_str_array
   module procedure fwrite_real, fwrite_real_array
   module procedure fwrite_double, fwrite_double_array
   module procedure fwrite_int, fwrite_int_array
end interface
```

Interface for fwrite.

```
interface associated
   module procedure fassociated
end interface
```

Function to do a fp!=NULL equivalent.

4.6.2 Subroutines and Functions

```
logical function feof(stream, errno)
  type(FILE), intent(in) :: stream
  integer, intent(out), optional :: errno
  logical, external :: c_feof
```

Determine if stream is at eof. See man 3 feof.

```
subroutine rewind(stream, errno)
  type(FILE), intent(inout) :: stream
  integer, intent(out), optional :: errno
```

Rewind a stream. See man 3 rewind.

```
subroutine fseek(stream, offset, whence, errno)
  type(FILE), intent(inout) :: stream
  integer(long_kind), intent(in) :: offset
  integer, intent(in), optional :: whence
  integer, intent(out), optional :: errno
```

Seek a stream at a position. See man 3 fseek

```
integer(long_kind) function ftell(stream, errno)
  type(FILE), intent(inout) :: stream
  integer, intent(out), optional :: errno
```

Get stream position. See man 3 ftell.

```
logical function fassociated(fp)
  type(FILE), intent(in) :: fp
```

Check that a stream is non-NULL.

```
type(FILE) function fopen(path, mode, errno)
  character(len=*), intent(in) :: path, mode
  integer, intent(out), optional :: errno
```

Open a stream. Use associated(FILE) to check the result is non-NULL. See man 3 fopen.

```
subroutine fclose(fp, errno)
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

Close a stream. See man 3 fclose.

```
type(FILE) function popen(command, mode, errno)
  character(len=*), intent(in) :: command, mode
  integer, intent(out), optional :: errno
```

Open a pipe to a command. See man 3 popen. Streams opened with popen must be pclosed.

```
subroutine pclose(fp, errno)
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

Close a stream opened with popen. See man 3 pclose.

```
integer(sizet_kind) function fread_str(str, length, fp, errno)
  character(len=*), intent(out) :: str
  integer(sizet_kind), intent(in) :: length
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fread() interface.

```
integer(sizet_kind) function fread_str_array(str, length, fp, errno)
  character(len=*), intent(out) :: str(:)
  integer(sizet_kind), intent(in) :: length
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fread() interface.

```
integer(sizet_kind) function fread_real(r, fp, errno)
  real, intent(out) :: r
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fread() interface.

```
integer(sizet_kind) function fread_real_array(r, fp, errno)
  real, intent(out) :: r(:)
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fread() interface.

```
integer(sizet_kind) function fread_double(d, fp, errno)
  real(double), intent(out) :: d
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fread() interface.

```
integer(sizet_kind) function fread_double_array(d, fp, errno)
  real(double), intent(out) :: d(:)
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fread() interface.

```
integer(sizet_kind) function fread_int(i, fp, errno)
  integer, intent(out) :: i
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fread() interface.

```
integer(sizet_kind) function fread_int_array(i, fp, errno)
  integer, intent(out) :: i(:)
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fread() interface.

```
integer(sizet_kind) function fwrite_str(str, length, fp, errno)
  character(len=*), intent(in) :: str
  integer(sizet_kind), intent(in) :: length
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fwrite() interface.

```
integer(sizet_kind) function fwrite_str_array(str, length, fp, errno)
  character(len=*), intent(in) :: str(:)
  integer(sizet_kind), intent(in) :: length
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fwrite() interface.

```
integer(sizet_kind) function fwrite_real(r, fp, errno)
  real, intent(in) :: r
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fwrite() interface.

```
integer(sizet_kind) function fwrite_real_array(r, fp, errno)
  real, intent(in) :: r(:)
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fwrite() interface.

```
integer(sizet_kind) function fwrite_double(r, fp, errno)
  real(double), intent(in) :: r
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fwrite() interface.

```
integer(sizet_kind) function fwrite_double_array(r, fp, errno)
  real(double), intent(in) :: r(:)
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fwrite() interface.

```
integer(sizet_kind) function fwrite_int(i, fp, errno)
  integer, intent(in) :: i
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fwrite() interface.

```
integer(sizet_kind) function fwrite_int_array(i, fp, errno)
  integer, intent(in) :: i(:)
  type(FILE), intent(inout) :: fp
  integer, intent(out), optional :: errno
```

One form of the fwrite() interface.

```
subroutine fgets(str, strlen, fp, errno)
  character(len=*), intent(inout) :: str
  integer, intent(out) :: strlen
  type(FILE), intent(in) :: fp
  integer, intent(out), optional :: errno
```

See man 3 fgets.

```
subroutine fputs(str, fp, errno)
  character(len=*), intent(in) :: str
  type(FILE):: fp
  integer, intent(out), optional :: errno
```

See man 3 fputs.

```
type(FILE) function stdin()
```

Get stream stdin.

```
type(FILE) function stdout()
```

Get stream stdout.

```
type(FILE) function stderr()
```

Get stream stderr.

4.7 Module f90_unix_proc

4.8 Module f90_unix_regexp

4.8.1 Parameters and Types

```
integer, parameter :: REG_EXTENDED
integer, parameter :: REG_ICASE
integer, parameter :: REG_NOSUB
integer, parameter :: REG_NEWLINE
integer, parameter :: REG_NOTBOL
integer, parameter :: REG_NOTEOL
```

Flags for regcomp and regexec. See man 3 regcomp

```
integer, parameter :: REGEX_KIND
integer, parameter :: REGMATCH_KIND
integer, parameter :: REGOFF_KIND
```

Kinds used in the module. You should not need to use these direcly.

```
type regex_t
   integer(regex_kind) :: rp
end type regex_t
```

Type used to hold compiled regular expressions. You should not need to access its components directly.

```
type regmatch_t
   integer(regoff_kind) :: rm_so, rm_eo
end type regmatch_t
```

Type used to hold indices of matches. Note that these indices are adjusted to fortran numbering.

4.8.2 Subroutines and Functions

```
subroutine regcomp(preg, regex, cflags, errc)
  type(regex_t) :: preg
  character(len=*), intent(in) :: regex
  integer, intent(in) :: cflags
  integer, intent(out), optional :: errc
```

See man 3 regcomp. Missing errc argument aborts on error.

```
subroutine regexec(preg, string, pmatch, eflags, errc)
  type(regex_t), intent(in) :: preg
  character(len=*), intent(in) :: string
  type(regmatch_t) :: pmatch(:)
  integer, intent(in) :: eflags
  integer, intent(out), optional :: errc
```

See man 3 regexec. Note that the offsets in pmatch are adjusted to fortran conventions, aka the first char has index 1, not 0. Missing errc argument aborts on error. No match is considered an error.

```
subroutine regerror(e, preg, msg)
  integer, intent(in) :: e
  type(regex_t), intent(in) :: preg
  character(len=*), intent(out) :: msg
```

See man 3 regerror.

```
subroutine regfree(preg)
  type(regex_t), intent(in) :: preg
```

See man 3 regfree.

4.9 Module f90_unix_signal

4.9.1 Introduction to fortran signal handling

Signal handling in Fortran faces an obstacle: The system calls the signal handler passing it an integer, whereas fortran expects a pointer to an integer. The library adresses this problem by installing a wrapper instead which accepts an integer and calls the fortran signal handler passing it a pointer to an integer. This is completly transparent to the user, except fot the special case where the signal handler is written in C. In this case, the SA_NOWRAPPER flag must be used which prevents installation of the wrapper. Note that sigaction passes back (in oldaction) the actual handler, not the wrapper. The SA_NOWRAPPER flag is automatically set if an action requires no wrapper.

4.9.2 Parameters and Types

```
integer, parameter :: funcp_kind
```

Integer kind to hold a function adress

```
integer, parameter :: SIG_IGN
integer, parameter :: SIG_DFL
```

Action codes to ignore a signal or use the default action.

```
integer, parameter :: SIGHUP
integer, parameter :: SIGINT
integer, parameter :: SIGQUIT
integer, parameter :: SIGILL
```

```
integer, parameter :: SIGTRAP
integer, parameter :: SIGABRT
integer, parameter :: SIGIOT
integer, parameter :: SIGBUS
integer, parameter :: SIGFPE
integer, parameter :: SIGKILL
integer, parameter :: SIGUSR1
integer, parameter :: SIGSEGV
integer, parameter :: SIGUSR2
integer, parameter :: SIGPIPE
integer, parameter :: SIGALRM
integer, parameter :: SIGTERM
integer, parameter :: SIGSTKFLT
integer, parameter :: SIGCLD
integer, parameter :: SIGCHLD
integer, parameter :: SIGCONT
integer, parameter :: SIGSTOP
integer, parameter :: SIGTSTP
integer, parameter :: SIGTTIN
integer, parameter :: SIGTTOU
integer, parameter :: SIGURG
integer, parameter :: SIGXCPU
integer, parameter :: SIGXFSZ
integer, parameter :: SIGVTALRM
integer, parameter :: SIGPROF
integer, parameter :: SIGWINCH
integer, parameter :: SIGPOLL
integer, parameter :: SIGIO
integer, parameter :: SIGPWR
integer, parameter :: SIGSYS
integer, parameter :: SIGUNUSED
integer, parameter :: NSIG
```

Signal codes. See man 7 signal

```
integer, parameter :: SA_NOCLDSTOP
integer, parameter :: SA_NOCLDWAIT
integer, parameter :: SA_RESETHAND
integer, parameter :: SA_ONSTACK
integer, parameter :: SA_RESTART
integer, parameter :: SA_NODEFER
integer, parameter :: SA_NOWRAPPER
integer, parameter :: SIG_BLOCK
integer, parameter :: SIG_UNBLOCK
integer, parameter :: SIG_SETMASK
```

Flags and other constants for sigaction.

Note that SA_SIGINFO is intentionally undefined- it cannot be used from within posix90.

```
type sigset_type
  character(len=128) :: sigset
end type
```

Type used to hold signal sets. See man 3 sigsetops.

```
type sigaction_type
   integer(funcp_kind) :: sa_handler
   type(sigset_type) :: sa_mask
   integer :: sa_flags
end type sigaction_type
```

Type to hold signal actions. Constructed using sigaction_compile. See sigaction.

```
interface sigaction_compile
   module procedure sigaction_compile_handler
   module procedure sigaction_compile_integer
end interface
```

Interface to construct a sigaction_type.

```
type(sigaction_type) function sigaction_compile_handler(handler, mask, flags, errno)
  interface
     subroutine handler(sig)
       integer, intent(in) :: sig
     end subroutine handler
  end interface
  type(sigset_type), intent(in), optional :: mask
  integer, intent(in), optional :: flags
  integer, intent(out), optional :: errno
```

Construct a sigaction_type. mask defaults to sigemptyset, flags to 0. Note the interface sigaction_compile; this routine should not be called directly.

```
type(sigaction_type) function sigaction_compile_integer(action_code, mask, flags, errno)
  integer, intent(in) :: action_code
  type(sigset_type), intent(in), optional :: mask
  integer, intent(in), optional :: flags
  integer, intent(out), optional :: errno
```

Construct a sigaction_type like action = sigaction_compile(SIG_IGN). Note the interface sigaction_compile. Errno EINVAL is set if action_code is neither SIG_IGN nor SIG_DFL.

```
subroutine sigaction(signum, action, oldaction, errno)
  integer :: signum
  type(sigaction_type), intent(in), optional :: action
  type(sigaction_type), intent(out), optional :: oldaction
  integer, intent(out), optional :: errno
```

See man 2 sigaction.

```
subroutine sigemptyset(set)
  type(sigset_type), intent(in):: set
```

See man 3 sigsetops.

```
subroutine sigfillset(set)
  type(sigset_type), intent(in):: set
```

See man 3 sigsetops.

```
subroutine sigaddset(set, sig1, sig2, sig3, sig4, sig5, sig6, sig7, sig8)
  type(sigset_type), intent(in):: set
  integer, intent(in) :: sig1
  integer, intent(in), optional :: sig2, sig3, sig4, sig5, sig6, sig7, sig8
```

See man 3 sigsetops.

```
subroutine sigdelset(set, sig1, sig2, sig3, sig4, sig5, sig6, sig7, sig8)
  type(sigset_type), intent(in):: set
  integer, intent(in) :: sig1
  integer, intent(in), optional :: sig2, sig3, sig4, sig5, sig6, sig7, sig8
```

See man 3 sigsetops.

```
logical function sigismember(set, sig)
  type(sigset_type), intent(in):: set
  integer, intent(in) :: sig
```

See man 3 sigsetops.

```
subroutine kill(pid, sig, errno)
  integer(pid_kind), intent(in) :: pid
  integer, intent(in) :: sig
  integer, intent(out), optional :: errno
```

See man 2 kill.

```
subroutine raise(sig,errno)
  integer, intent(in) :: sig
  integer, intent(out), optional :: errno
```

See man 3 raise.

4.10 Module f90_unix_time

4.10.1 Parameters and Types

```
integer, parameter :: TIME_KIND
```

integer kind equivalent to C time_t.

4.10.2 Subroutines and Functions

```
character(len=30) function ctime(time)
  integer(time_kind), intent(in) :: time
```

See man 3 ctime.

Appendix A GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.
4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year  name of author

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., 51 Franklin Street, Fifth Floor,
Boston, MA 02110-1301, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Concept Index

G

I

K

L

M

N

O

P

R

S

T

U

W

X

Short Contents

Table of Contents