

f90_unix_proc: UNIX Process Primitives Module

March 15, 2019

1 Name

f90_unix_proc — Module of Unix process primitives

2 Usage

USE F90_UNIX_PROC

This module contains part of a Fortran API to functions detailed in ISO/IEC 9945-1:1990 Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language].

The functions in this module are from Section 3: Process Primitives, excluding 3.3 Signals. The C language functions **abort**, **atexit**, **exit** and **system** are also provided by this module.

Error handling is described in **F90_UNIX_ERRNO**. Note that for procedures with an optional **ERRNO** argument, if an error occurs and **ERRNO** is not present, the program will be terminated.

All the procedures in this module are generic; some may be specific but this is subject to change in a future release.

3 Synopsis

Parameters

ATOMIC_INT, **ATOMIC_LOG**, **PID_KIND** (renamed **ID_KIND** from **F90_UNIX_ENV**, **TIME_KIND** (from **F90_UNIX_ENV**), **WNOHANG**, **WUNTRACED**.

Generic Procedures

ABORT, **ALARM**, **ATEXIT**, **EXECL**, **EXECLP**, **EXECV**, **EXECVE**, **EXECVP**, **EXIT**, **FASTEXIT**, **FORK**, **PAUSE**, **SLEEP**, **SYSTEM**, **WAIT**, **WAITPID**, **WEXITSTATUS**, **WIFEXITED**, **WIFSIGNALED**, **WIFSTOPPED**, **WSTOPSIG**, **WTERMSIG**.

Note that some of the generic procedures might be specific; this could change in a future release.

4 Parameter Description

INTEGER(int32),PARAMETER :: atomic_int

Integer kind for “atomic” operations in signal handlers (see **ALARM**).

INTEGER(int32),PARAMETER :: atomic_log

Logical kind for “atomic” operations in signal handlers (see **ALARM**).

USE f90_unix_env, ONLY: pid_kind=>id_kind

Integer kind for representing process IDs; this has been superseded by **ID_KIND** from **F90_UNIX_ENV**.

USE f90_unix_env, ONLY: time_kind

Integer kind for representing times in seconds.

```
INTEGER(int32),PARAMETER :: wnohang
```

Option bit for WAITPID indicating that the calling process should not wait for the child process to stop or exit.

```
INTEGER(int32),PARAMETER :: wuntraced
```

Option bit for WAITPID indicating that status should be returned for stopped processes as well as terminated ones.

5 Procedure Description

```
SUBROUTINE abort(message)
  CHARACTER(*),OPTIONAL :: message
```

ABORT cleans up the i/o buffers and then terminates execution, producing a core dump on Unix systems. If MESSAGE is given it is written to logical unit 0 (zero) preceded by ‘ abort:’.

```
SUBROUTINE alarm(seconds,subroutine,seclft,errno)
  INTEGER(*),INTENT(IN) :: seconds
  INTERFACE
    SUBROUTINE subroutine()
  END
  END INTERFACE
  OPTIONAL subroutine
  INTEGER(time_kind),OPTIONAL,INTENT(OUT) :: seclft
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Establishes an “alarm” call to the procedure SUBROUTINE to occur after SECONDS seconds have passed, or cancels an existing alarm if SECONDS==0. If SUBROUTINE is not present, any previous association of a subroutine with the alarm signal is left unchanged. If SECLFT is present, it receives the number of seconds that were left on the preceding alarm or zero if there were no existing alarm.

The subroutine invoked by the alarm call is only permitted to define VOLATILE SAVED variables that have the type INTEGER(atomic_int) or LOGICAL(atomic_log); defining or referencing any other kind of variable may result in unpredictable behaviour, even program termination. Furthermore, it shall not perform any array or CHARACTER operations, input/output, or invoke any intrinsic function or module procedure.

If an alarm call is established with no handler (i.e. SUBROUTINE was not present on the first call) the process may be terminated when the alarm goes off.

Possible errors include ENOSYS.

```
SUBROUTINE atexit(subroutine,errno)
  INTERFACE
    SUBROUTINE subroutine()
  END
  END INTERFACE
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Registers an argumentless subroutine for execution on normal termination of the program.

If the program terminates normally, all subroutines registered with ATEXTIT will be invoked in reverse order of their registration. Normal termination includes using the F90_UNIX.PROC procedure EXIT, executing a Fortran STOP statement or executing a main program END statement. ATEXTIT subroutines are invoked before automatic file closure.

If the program terminates due to an error or by using the F90_UNIX.PROC procedure FASTEXIT, these subroutines will not be invoked.

Possible errors include ENOMEM.

Note: The list of ATEXIT procedures registered via Fortran is separate from those registered via C; the latter will be invoked after all files have been closed.

```
SUBROUTINE execl(path,arg0...,errno)
  CHARACTER(*),INTENT(IN) :: path
  CHARACTER(*),INTENT(IN) :: arg0...
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Executes a file instead of the current image, like EXECV. The arguments to the new program are specified by the dummy arguments which are named ARG0, ARG1, etc. up to ARG20 (additional arguments may be provided in later releases). Note that these are not optional arguments, any actual argument that is itself an optional dummy argument must be present. This function is the same as EXECV except that the arguments are provided individually instead of via an array; and because they are provided individually, there is no need to provide the lengths (the lengths being taken from each argument itself).

Errors are the same as for EXECV.

```
SUBROUTINE execlp(file,arguments,,errno)
  CHARACTER(*),INTENT(IN) :: file
  CHARACTER(*),INTENT(IN) :: arguments
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Executes a file instead of the current image, like EXECV. The arguments to the new program are specified by the dummy arguments which are named ARG0, ARG1, etc. up to ARG20 (additional arguments may be provided in later releases). Note that these are not optional arguments, any actual argument that is itself an optional dummy argument must be present. This function is the same as EXECL except that determination of the program to be executed follows the same rules as EXECVP.

Errors are the same as for EXECV.

```
SUBROUTINE execv(path,argv,lenargv,errno)
  CHARACTER(*),INTENT(IN) :: path
  CHARACTER(*),INTENT(IN) :: argv(:)
  INTEGER(*),INTENT(IN) :: lenargv(:)
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Executes the file PATH in place of the current process image; for full details see ISO/IEC 9945-1:1990 section 3.1.2. ARGV is the array of argument strings, LENARGV containing the desired length of each argument. If ARGV is not zero-sized, ARGV(1):(LENARGV(1)) is passed as argument zero (i.e. the program name).

If LENARGV is not the same shape as ARGV, error EINVAL is raised (see F90_UNIX_ERRNO). Other possible errors include E2BIG, EACCES, ENAMETOOLONG, ENOENT, ENOTDIR, ENOEXEC and ENOMEM.

```
SUBROUTINE EXECVE(path,argv,lenargv,env,lenenv,errno)
  CHARACTER(*),INTENT(IN) :: path
  CHARACTER(*),INTENT(IN) :: argv(:)
  INTEGER(*),INTENT(IN) :: lenargv(:)
  CHARACTER(*),INTENT(IN) :: env(:)
  INTEGER(*),INTENT(IN) :: lenenv(:)
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Similar to EXECV, with the environment strings specified by ENV and LENENV being passed to the new program; for full details see ISO/IEC 9945-1:1990 section 3.1.2.

If LENARGV is not the same shape as ARGV or LENENV is not the same shape as ENV, error EINVAL is raised (see F90_UNIX_ERRNO). Other errors are the same as for EXECV.

```

SUBROUTINE execvp(file,argv,lenargv,errno)
  CHARACTER(*),INTENT(IN) :: file
  CHARACTER(*),INTENT(IN) :: argv(:)
  INTEGER(*),INTENT(IN) :: lenargv(:)
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno

```

The same as EXECV except that the program to be executed, FILE, is searched for using the PATH environment variable (unless it contains a slash character, in which case EXECVP is identical in effect to EXECV).

Errors are the same as for EXECV.

```

SUBROUTINE exit(status)
  INTEGER(int32),OPTIONAL,INTENT(IN) :: status

```

Terminate execution as if executing the END statement of the main program (or an unadorned STOP statement). If STATUS is given it is returned to the operating system (where applicable) as the execution status code.

```

SUBROUTINE fastexit(status)
  INTEGER,OPTIONAL,INTENT(IN) :: status

```

This provides the functionality of ISO/IEC 9945-1:1990 function `_exit` (section 3.2.2). There are two main differences between FASTEXIT and EXIT:

1. When EXIT is called all open logical units are closed (as if by a CLOSE statement). With FASTEXIT this is not done, nor are any file buffers flushed, thus the contents and status of any file connected at the time of calling FASTEXIT are undefined.
2. Subroutines registered with ATEXTIT are not executed.

```

SUBROUTINE fork(pid,errno)
  INTEGER(id_kind),INTENT(OUT) :: pid
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno

```

Creates a new process which is an exact copy of the calling process. In the new process, the value returned in PID is zero; in the calling process the value returned in PID is the process ID of the new (child) process.

Possible errors include EAGAIN, ENOMEM and ENOSYS (see F90_UNIX_ERRNO).

```

SUBROUTINE pause(errno)
  INTEGER(error_kind),INTENT(OUT) :: errno

```

Suspends process execution until a signal is raised. If the action of the signal was to terminate the process, the process is terminated without returning from PAUSE. If the action of the signal was to invoke a signal handler (e.g. via ALARM), process execution continues after return from the signal handler.

If process execution is continued after a signal, ERRNO is set to EINTR.

If this functionality is not available, ERRNO is set to ENOSYS.

```

PURE SUBROUTINE sleep(seconds,seclft)
  INTEGER(*),INTENT(IN) :: seconds
  INTEGER(time_kind),OPTIONAL,INTENT(OUT) :: seclft

```

Suspends process execution for SECONDS seconds, or until a signal has been delivered. If SECLFT is present, it receives the number of seconds remaining in the sleep time (zero unless the sleep was interrupted by a signal).

```

SUBROUTINE system(string,status,errno)
  CHARACTER(*),INTENT(IN) :: string
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: status,errno

```

Passes STRING to the command processor for execution. If STATUS is present it receives the completion status - this is the same status returned by WAIT and can be decoded with WIFEXITED etc. If ERRNO is present it receives the error status from the SYSTEM call itself.

Possible errors are those from FORK or EXECV.

```

SUBROUTINE wait(status,retpid,errno)
  INTEGER(int32),OPTIONAL,INTENT(OUT) :: status
  INTEGER(id_kind),OPTIONAL,INTENT(OUT) :: retpid
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno

```

Wait for any child process to terminate (returns immediately if one has already terminated). See ISO/IEC 9945-1:1990 section 3.2.1 for full details.

If STATUS is present it receives the termination status of the child process. If RETPID is present it receives the process number of the child process.

Possible errors include ECHILD and EINTR (see F90_UNIX_ERRNO).

```

SUBROUTINE waitpid(pid,status,options,retpid,errno)
  INTEGER(id_kind),INTENT(IN) :: pid
  INTEGER(int32),OPTIONAL,INTENT(OUT) :: status
  INTEGER(int32),OPTIONAL,INTENT(IN) :: options
  INTEGER(id_kind),OPTIONAL,INTENT(OUT) :: retpid
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno

```

Wait for a particular child process to terminate (or for any one if PID==(-1)). If OPTIONS is not present it is as if it were present with a value of 0. See ISO/IEC 9945-1:1990 section 3.2.1 for full details.

Possible errors include ECHILD, EINTR and EINVAL (see F90_UNIX_ERRNO).

```

PURE INTEGER(int32) FUNCTION wexitstatus(stat_val)
  INTEGER(int32),INTENT(IN) :: stat_val

```

If WIFEXITED(STAT_VAL) is .TRUE., this function returns the low-order 8 bits of the status value supplied to EXIT or FASTEXIT by the child process. If the child process executed a STOP statement or main program END statement, the value will be zero. If WIFEXITED(STAT_VAL) is .FALSE., the function value is undefined.

```

PURE LOGICAL(word) FUNCTION wifexited(stat_val)
  INTEGER(error_kind),INTENT(IN) :: stat_val

```

Returns .TRUE. if and only if the child process terminated by calling FASTEXIT, EXIT, by executing a STOP statement or main program END statement, or possibly by some means other than Fortran.

```

PURE LOGICAL(word) FUNCTION wifsignaled(stat_val)
  INTEGER(int32),INTENT(IN) :: stat_val

```

Returns .TRUE. if and only if the child process terminated by receiving a signal that was not caught.

```

PURE LOGICAL(word) FUNCTION wifstopped(stat_val)
  INTEGER(int32),INTENT(IN) :: stat_val

```

Returns `.TRUE.` if and only if the child process is stopped (and not terminated). Note that `WAITPID` must have been used with the `WUNTRACED` option to receive such a status value.

```
PURE INTEGER(int32) FUNCTION wstopsig(stat_val)
    INTEGER(int32), INTENT(IN) :: stat_val
```

If `WIFSTOPPED(STAT_VAL)` is `.TRUE.`, this function returns the signal number that caused the child process to stop. If `WIFSTOPPED(STAT_VAL)` is `.FALSE.`, the function value is undefined.

```
PURE INTEGER(int32) FUNCTION wtermsig(stat_val)
    INTEGER(int32), INTENT(IN) :: stat_val
```

If `WIFSIGNALED(STAT_VAL)` is `.TRUE.`, this function returns the signal number that caused the child process to terminate. If `WIFSIGNALED(STAT_VAL)` is `.FALSE.`, the function value is undefined.

6 See Also

`f90_unix_errno(3)`, `intro(3)`, `nag_modules(3)`, `nagfor(1)`.

7 Bugs

Please report any bugs found to ‘support@nag.co.uk’ or ‘support@nag.com’, along with any suggestions for improvements.