

Cygwin API Reference

DJ Delorie

Geoffrey Noer

Cygwin API Reference

by DJ Delorie and Geoffrey Noer

Copyright © 1998, 1999, 2000, 2001, 2002, 2003 Red Hat, Inc.

Revision History

Revision 0.0 1998-08-31 Revised by: dj

Initial revision

Revision 0.5.0 1998-12-17 Revised by: noer

Add pthread, sem calls. Change revnumber to three-part number: Cygwin API major, Cygwin API minor, Doc rev number. Starts out at

Table of Contents

1. Compatibility	1
1.1. Compatibility with ANSI.....	1
1.1.1. stdio	1
1.1.2. string	1
1.1.3. stdlib	1
1.1.4. time	1
1.1.5. signals.....	1
1.1.6. ctype	1
1.1.7. math	1
1.1.8. misc.....	2
1.2. Compatibility with POSIX.1.....	2
1.2.1. Process Primitives (Section 3).....	2
1.2.2. Process Environment (Section 4)	2
1.2.3. Files and Directories (Section 5).....	2
1.2.4. Input and Output Primitives (Section 6).....	2
1.2.5. Device- and Class-Specific Functions (Section 7)	2
1.2.6. Language-Specific Services for the C Programming Language (Section 8).....	3
1.2.7. System Databases (Section 9)	3
1.2.8. Synchronization (Section 11)	3
1.2.9. Memory Management (Section 12).....	3
1.2.10. Thread Management (Section 16).....	3
1.2.11. Thread-Specific Data Functions (Section 17).....	3
1.2.12. Implementation Details	3
1.3. Compatibility with Miscellaneous Other Standards	4
1.3.1. Networking	4
1.3.2. Other	4
1.3.3. Implementation Notes	5
2. Cygwin Functions	6
2.1. cygwin_attach_handle_to_fd	6
2.2. cygwin_conv_to_full_posix_path	6
2.3. cygwin_conv_to_full_win32_path.....	6
2.4. cygwin_conv_to_posix_path.....	6
2.5. cygwin_conv_to_win32_path	7
2.6. cygwin_detach_dll	7
2.7. cygwin_getshared.....	7
2.8. cygwin_internal.....	7
2.9. cygwin_posix_path_list_p.....	8
2.10. cygwin_posix_to_win32_path_list	8
2.11. cygwin_posix_to_win32_path_list_buf_size	9
2.12. cygwin_split_path	9
2.13. cygwin_stackdump.....	9
2.14. cygwin_win32_to_posix_path_list	10
2.15. cygwin_win32_to_posix_path_list_buf_size	10
2.16. cygwin_winpid_to_pid.....	10

List of Examples

2-1. Example use of <code>cygwin_posix_to_win32_path_list</code>	8
2-2. Example use of <code>cygwin_split_path</code>	9
2-3. Example use of <code>cygwin_winpid_to_pid</code>	10

Chapter 1. Compatibility

1.1. Compatibility with ANSI

The following functions are compatible with ANSI:

1.1.1. stdio

clearerr, fclose, feof, ferror, fflush, fgetc, fgetpos, fgets, fopen, fprintf, fputc, fputs, fread, freopen, fscanf, fseek, fsetpos, ftell, fwrite, getc, getchar, gets, perror, printf, putc, putchar, puts, remove, rename, rewind, scanf, setbuf, setvbuf, sprintf, sscanf, tmpfile, tmpnam, vfprintf, ungetc, vprintf, vsprintf,

1.1.2. string

memchr, memcmp, memcpy, memmove, memset, strcat, strchr, strcmp, strcoll, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strchr, strspn, strstr, strtok, strxfrm

1.1.3. stdlib

abort, abs, assert, atexit, atof, atoi, atol, bsearch, calloc, div, exit, free, getenv, labs, ldiv, longjmp, malloc, mblen, mbstowcs, mbtowc, qsort, rand, realloc, setjmp, srand, strtod, strtol, strtoul, system, wcstombs, wctomb

1.1.4. time

asctime, gmtime, localtime, time, clock, ctime, difftime, mktime, strftime

1.1.5. signals

raise, signal

1.1.6. ctype

isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper

1.1.7. math

acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan, tanh

1.1.8. misc

localeconv, setlocale, va_arg, va_end, va_start

1.2. Compatibility with POSIX.1

The following functions are compatible with POSIX.1:

1.2.1. Process Primitives (Section 3)

fork, execl, execl, execlp, execv, execve, execvp, wait, waitpid, _exit, kill, sigemptyset, sigfillset, sigaddset, sigdelset, sigismember, sigaction, pthread_sigmask, sigprocmask, sigpending, sigsuspend, alarm, pause, sleep, pthread_kill, pthread_sigmask

1.2.2. Process Environment (Section 4)

getpid, getppid, getuid, geteuid, getgid, getegid, setuid, setgid, getgroups, getlogin, getpgrp, setsid, setpgid, uname, time, times, getenv, ctermid, ttyname, isatty, sysconf

1.2.3. Files and Directories (Section 5)

opendir, readdir, rewinddir, closedir, chdir, getcwd, open, creat, umask, link, mkdir, unlink, rmdir, rename, stat, fstat, access, chmod, fchmod, chown, utime, ftruncate, pathconf, fpathconf

1.2.4. Input and Output Primitives (Section 6)

pipe, dup, dup2, close, read, write, fcntl, lseek, fsync

1.2.5. Device- and Class-Specific Functions (Section 7)

cfgetispeed, cfgetospeed, cfsetispeed, cfsetospeed, tcdrain, tcflow, tcflush, tcgetattr, tcgetpgrp, tcseendbreak, tcsetattr, tcsetpgrp

1.2.6. Language-Specific Services for the C Programming Language (Section 8)

abort, exit, fclose, fdopen, fflush, fgetc, fgets, fileno, fopen, fprintf, fputc, fputs, fread, freopen, fscanf, fseek, ftell, fwrite, getc, getchar, gets, perror, printf, putc, putchar, puts, remove, rewind, scanf, setlocale, siglongjmp, sigsetjmp, tmpfile, tmpnam, tzset

1.2.7. System Databases (Section 9)

getgrgid, getgrnam, getpwnam, getpwuid

1.2.8. Synchronization (Section 11)

sem_init, sem_destroy, sem_wait, sem_trywait, sem_post, pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock

1.2.9. Memory Management (Section 12)

mmap, mprotect, msync, munmap

1.2.10. Thread Management (Section 16)

pthread_attr_init, pthread_attr_destroy, pthread_attr_setstacksize, pthread_attr_getstacksize, pthread_create, pthread_exit, pthread_self, pthread_equal

1.2.11. Thread-Specific Data Functions (Section 17)

pthread_key_create, pthread_setspecific, pthread_getspecific, pthread_key_delete

1.2.12. Implementation Details

`setuid` and `setgid` always return `ENOSYS`.

`link` will copy the file if it can't implement a true symbolic link. Currently, symbolic links work, if at all, only under Windows NT.

`chown` always returns zero.

`fcntl` doesn't support `F_GETLK` - it returns -1 and sets `errno` to `ENOSYS`.

`lseek` only works properly on binary files.

1.3. Compatibility with Miscellaneous Other Standards

The following functions are compatible with miscellaneous other standards:

1.3.1. Networking

(Standardized by POSIX 1.g, which is probably still in draft?)

`accept`, `bind`, `connect`, `getdomainname`, `gethostbyaddr`, `gethostbyname`, `getpeername`, `getprotobyname`, `getprotobynumber`, `getservbyname`, `getservbyport`, `getsockname`, `getsockopt`, `herror`, `htonl`, `htons`, `inet_addr`, `inet_makeaddr`, `inet_netof`, `inet_ntoa`, `listen`, `ntohl`, `ntohs`, `rcmd`, `recv`, `recvfrom`, `rexec`, `rresvport`, `send`, `sendto`, `setsockopt`, `shutdown`, `socket`, `socketpair`

Of these networking calls, `rexec`, `rcmd` and `rresvport` are implemented in MS IP stack but may not be implemented in other vendors' stacks.

1.3.2. Other

`chroot`, `closelog`, `cwait`, `dlclose`, `dlderror`, `dlfork`, `dlopen`, `dlsym`, `endgrent`, `ffs`, `fstatfs`, `ftime`, `get_osfhandle`, `getdtablesize`, `getgrent`, `gethostname`, `getitimer`, `getmntent`, `getpagesize`, `getpgid`, `getpwent`, `gettimeofday`, `grantpt`, `initgroups`, `ioctl`, `killpg`, `login`, `logout`, `lstat`, `mknod`, `memccpy`, `nice`, `openlog`, `pclose`, `popen`, `ptsname`, `putenv`, `random`, `readv`, `realpath`, `regfree`, `rexec`, `select`, `setegid`, `setenv`, `seterrno`, `seteuid`, `setitimer`, `setmntent`, `setmode`, `setpassent`, `setpgrp`, `setpwent`, `settimeofday`, `sexec1`, `sexecle`, `sexec1p`, `sexec1pe`, `sexecp`, `sexecv`, `sexecve`, `sexecvpe`, `sigpause`, `spawnl`, `spawnle`, `spawnlp`, `spawnlpe`, `spawnv`, `spawnve`, `spawnvp`, `spawnvpe`, `srandom`, `statfs`, `strsignal`, `strtosigno`, `swab`, `syslog`, `timezone`,

truncate, ttyslot, unlockpt, unsetenv, usleep, utimes, vfork, vhangup, wait3, wait4, wscmp, wcslen, wprintf, writev

1.3.3. Implementation Notes

`initgroups` does nothing

`chroot`, `mknod`, `settimeofday`, and `vhangup` always return -1 and sets `errno` to `ENOSYS`.

`nice` allows Cygwin programs to alter their current runtime priority through the use of its `incr` argument. Cygwin processes can be set to `IDLE_PRIORITY_CLASS`, `NORMAL_PRIORITY_CLASS`, `HIGH_PRIORITY_CLASS`, or `REALTIME_PRIORITY_CLASS` with the `nice` call. `NORMAL_PRIORITY_CLASS` is the default. If you pass a positive number to `nice()`, then the priority level will decrease by one (within the above list of priorities). A negative number would make it increase by one. It is not possible to change it by more than one at a time without making repeated calls. An increment above `REALTIME_PRIORITY_CLASS` results in the process staying at that priority. Likewise, a decrement to `IDLE_PRIORITY_CLASS` has it stay at that priority. Note that in the Win32 API, there are 32 priorities. So currently we only give access to four of these through `nice`.

`seteuid` and `setegid` always return 0 and set `errno` to `ENOSYS`.

`vfork` just calls `fork`

Chapter 2. Cygwin Functions

These functions are specific to Cygwin itself, and probably won't be found anywhere else.

2.1. `cygwin_attach_handle_to_fd`

```
extern "C" int cygwin_attach_handle_to_fd(char *name, int fd, HANDLE handle,
int bin, int access);
```

This function can be used to turn a Win32 "handle" into a posix-style file handle. *fd* may be -1 to make cygwin allocate a handle; the actual handle is returned in all cases.

2.2. `cygwin_conv_to_full_posix_path`

```
extern "C" void cygwin_conv_to_full_posix_path(const char *path, char
*posix_path);
```

Converts a Win32 path to a POSIX path. If *path* is already a POSIX path, leaves it alone. If *path* is relative, then *posix_path* will be converted to an absolute path. Note that *posix_path* must point to a buffer of sufficient size; use `MAX_PATH` if needed.

2.3. `cygwin_conv_to_full_win32_path`

```
extern "C" void cygwin_conv_to_full_win32_path(const char *path, char
*win32_path);
```

Converts a POSIX path to a Win32 path. If *path* is already a Win32 path, leaves it alone. If *path* is relative, then *win32_path* will be converted to an absolute path. Note that *win32_path* must point to a buffer of sufficient size; use `MAX_PATH` if needed.

2.4. `cygwin_conv_to_posix_path`

```
extern "C" void cygwin_conv_to_posix_path(const char *path, char
*posix_path);
```

Converts a Win32 path to a POSIX path. If *path* is already a POSIX path, leaves it alone. If *path* is relative, then *posix_path* will also be relative. Note that *posix_path* must point to a buffer of sufficient size; use `MAX_PATH` if needed.

2.5. `cygwin_conv_to_win32_path`

```
extern "C" void cygwin_conv_to_win32_path(const char *path, char
*win32_path);
```

Converts a POSIX path to a Win32 path. If *path* is already a Win32 path, leaves it alone. If *path* is relative, then *win32_path* will also be relative. Note that *win32_path* must point to a buffer of sufficient size; use `MAX_PATH` if needed.

2.6. `cygwin_detach_dll`

```
extern "C" void cygwin_detach_dll(int dll_index);
```

2.7. `cygwin_getshared`

```
shared_info * cygwin_getshared(void);
```

Returns a pointer to an internal Cygwin memory structure containing shared information used by cooperating cygwin processes. This function is intended for use only by "system" programs like `mount` and `ps`.

2.8. cygwin_internal

```
extern "C" DWORD cygwin_internal(cygwin_getinfo_types t, ...);
```

This function gives you access to various internal data and functions. It takes two arguments. The first argument is a type from the 'cygwin_getinfo_types' enum. The second is an optional pointer.

Stay away unless you know what you're doing.

2.9. cygwin_posix_path_list_p

```
extern "C" int posix_path_list_p(const char *path);
```

This function tells you if the supplied *path* is a POSIX-style path (i.e. posix names, forward slashes, colon delimiters) or a Win32-style path (drive letters, reverse slashes, semicolon delimiters). The return value is true if the path is a POSIX path. Note that "_p" means "predicate", a lisp term meaning that the function tells you something about the parameter.

Rather than use a mode to say what the "proper" path list format is, we allow any, and give apps the tools they need to convert between the two. If a ';' is present in the path list it's a Win32 path list. Otherwise, if the first path begins with [letter]: (in which case it can be the only element since if it wasn't a ';' would be present) it's a Win32 path list. Otherwise, it's a POSIX path list.

2.10. cygwin_posix_to_win32_path_list

```
extern "C" void cygwin_posix_to_win32_path_list(const char *posix, char *win32);
```

Given a POSIX path-style string (i.e. /foo:/bar) convert it to the equivalent Win32 path-style string (i.e. d:\;e:\bar). *win32* must point to a sufficiently large buffer.

Example 2-1. Example use of cygwin_posix_to_win32_path_list

```
char *_epath;
char *_win32epath;
_epath = _win32epath = getenv (NAME);
```

```

/* If we have a POSIX path list, convert to win32 path list */
if (_epath != NULL && *_epath != 0
    && cygwin_posix_path_list_p (_epath))
{
    _win32epath = (char *) xmalloc
        (cygwin_posix_to_win32_path_list_buf_size (_epath));
    cygwin_posix_to_win32_path_list (_epath, _win32epath);
}

```

See also `cygwin_posix_to_win32_path_list_buf_size`

2.11. `cygwin_posix_to_win32_path_list_buf_size`

```

extern "C" int cygwin_posix_to_win32_path_list_buf_size(const char
*path_list);

```

Returns the number of bytes needed to hold the result of calling `cygwin_posix_to_win32_path_list`.

2.12. `cygwin_split_path`

```

extern "C" void cygwin_split_path (const char * path, char * dir, char *
file);

```

Split a path into the directory and the file portions. Both *dir* and *file* are expected to point to buffers of sufficient size.

Example 2-2. Example use of `cygwin_split_path`

```

char dir[200], file[100];
cygwin_split_path("c:/foo/bar.c", dir, file);
printf("dir=%s, file=%s\n", dir, file);

```

2.13. `cygwin_stackdump`

```

extern "C" void cygwin_stackdump(void);

```

Outputs a stackdump to stderr from the called location.

2.14. `cygwin_win32_to_posix_path_list`

```
extern "C" void cygwin_win32_to_posix_path_list(const char *win32, char
*posix);
```

Given a Win32 path-style string (i.e. d:\e:\bar) convert it to the equivalent POSIX path-style string (i.e. /foo:/bar). *posix* must point to a sufficiently large buffer. See also `cygwin_win32_to_posix_path_list_buf_size`

2.15. `cygwin_win32_to_posix_path_list_buf_size`

```
extern "C" int cygwin_win32_to_posix_path_list_buf_size(const char
*path_list);
```

Tells you how many bytes are needed for the results of `cygwin_win32_to_posix_path_list`.

2.16. `cygwin_winpid_to_pid`

```
extern "C" pid_t cygwin_winpid_to_pid (int winpid);
```

Given a windows pid, converts to the corresponding Cygwin pid, if any. Returns -1 if windows pid does not correspond to a cygwin pid.

Example 2-3. Example use of `cygwin_winpid_to_pid`

```
extern "C" cygwin_winpid_to_pid (int winpid);
pid_t mypid;
mypid = cygwin_winpid_to_pid (windows_pid);
```