Os Library Reference

September 22, 2014

Contents

Description

Provides interface to operating system services through the C run-time library for the system.

Note: You may want to enable Previous View and Next View commands in the PDF viewer for comfortable browsing experience in this document.

Copyrights

Namespaces

Namespace	Description
Global	Functions in this library are put to the global namespace for compatibility with the C programming language.

1 Global Namespace

Functions in this library are put to the global namespace for compatibility with the C programming language.

1.1 Type Definitions

Name	Type	Description
cond_t	void*	Placeholder type for a platform specific condition variable handle type.
mutex_t	void*	Placeholder type for a platform specific mutex handle type.
thread_t	ulong	Placeholder type for a platform specific thread handle type.

1.2 Functions

Function	Description
allocate_cond_handle()	Allocates a condition variable handle.
allocate_mutex_handle()	Allocates a mutex handle.
allocate_thread_data(ulong)	Allocates thread specific data from the thread table: an exception table, a call stack and a frame pointer.
begin_capture_call_stack()	Locks a mutex that provides access to a global memory for a call stack string.
broadcast_cond(void**)	Unblocks all threads that wait on a condition variable.
capture_call_stack()	Formats a call stack string for the calling thread to a global call stack buffer and returns a pointer to it.
close(int)	Closes a file.
create_thread(ulong*, thread_fun, void*)	Creates a thread and starts executing the thread start function asynchronously.
dbgheap_done()	Frees memory allocated for the debug heap.
dbgheap_free(void*)	Frees a memory block allocated from the debug heap.
dbgheap_get_serial(void*)	Returns allocation serial number for a memory block allocated from the debug heap.
dbgheap_init()	Initializes the debug heap.
dbgheap_malloc(ulong, const char*, int)	Allocates a memory block from the debug heap.
dbgheap_report()	Prints a report of detected memory leaks.
dbgheap_stop(int)	Sets the stop serial number of an allocation. When dbgheap_malloc(ulong, const char*, int) function reaches the stop serial number, it prints the source file name and line number of that allocation.
destroy_cond(void**)	Destroyes a condition variable.

destroy_mutex(void**)	Destroyes a mutex.
detach_thread(ulong)	Detaches a thread.
directory_exists(const char*)	Returns 1 if a directory for the given path name exists, 0 otherwise.
end_capture_call_stack()	Unlocks a mutex that provides access to a global memory for a call stack string.
enter_frame(const char*, const char*, int)	Adds a call frame to the call stack of the calling thread.
equal_thread(ulong, ulong)	Returns true if the given thread identifiers refer to the same thread, false otherwise.
exit(int)	Terminates the calling process after doing a cleanup.
exit_thread(void*)	Exits the current thread.
file_exists(const char*)	Returns 1 if a file for the given path name exists, 0 otherwise.
free(void*)	Returns a block of memory back to the system.
free_cond_handle(void**)	Frees a condition variable handle.
free_mutex_handle(void**)	Frees a mutex handle.
free_thread_data(ulong)	Frees thread specific data from the thread table.
get_current_working_directory(char*, int)	Obtains a path to the current working directory for the calling process.
get_default_pmode()	Returns the default permission mode for the platform.

get_environment_variable(const char*)	Gets a value of an environment variable.
get_errno()	Returns the value of the C run-time library errno variable.
get_exception_table_addr(ulong)	Returns the address of the exception table for the given thread.
init_cond(void**)	Initializes a condition variable.
init_mutex(void**)	Initializes a mutex.
init_recursive_mutex(void**)	Initializes a recursive mutex.
join_thread(ulong, void**)	Waits for the given thread to terminate.
leave_frame()	Removes a call frame to the call stack of the calling thread.
lock_mutex(void**)	Locks a mutex. If the mutex is currently locked, waits for the mutex to become unlocked.
lseek(int, long, int)	Moves the file pointer to the specified location.
malloc(ulong)	Obtains a block of memory from the system.
mutextbl_done()	Destroyes mutexes in the mutex table and frees the memory allocated for the mutex table.
mutextbl_init(int)	Allocates memory for the mutex table for the given number of mutexes and initializes the mutexes.
mutextbl_lock_mtx(int)	Locks a mutex that has given number.
mutextbl_unlock_mtx(int)	Unlocks a mutex that has given number.
open_file(const char*, OpenFlags, int)	Opens a file with the specified name, mode and permissions.

path_exists(const char*)	Return 1 if a given path name exists, 0 otherwise.
rand()	Returns a pseudorandom integer.
read(int, void*, uint)	Reads data from a file.
signal_cond(void**)	Unblocks one thread waiting on a condition variable.
sleep(long, int)	Puts the calling thread to sleep for the given number of seconds and nanoseconds.
srand(uint)	Sets the seed value of the pseudorandom number generator.
strerror(int)	Gets a system error message for the given C run-time library error code (errno).
system(const char*)	Executes a command.
this_thread()	Returns the identifier of the calling thread.
threadtbl_done()	Frees the memory allocated for the thread table.
threadtbl_init(int)	Allocates memory for the thread table.
time_nanosecs(long&, int&)	Gets the current time from the real time clock of the computer.
timedwait_cond(void**, void**, long, int, bool&)	Waits for a condition variable to become signaled or given time to elapse.
trylock_mutex(void**, bool&)	Locks a mutex and sets a Boolean variable pointed by already_locked to 0, if the mutex is currently unlocked, otherwise sets a Boolean variable pointed by to 1.

unlock_mutex(void**)	Unlocks a mutex.
wait_cond(void**, void**)	Waits a condition variable to become signaled.
write(int, void*, uint)	Writes data to a file.

1.2.1 allocate_cond_handle() Function

Allocates a condition variable handle.

Syntax

```
public cdecl void** allocate_cond_handle();
```

Returns

void**

Returns pointer to a new uninitialized condition variable handle.

1.2.2 allocate_mutex_handle() Function

Allocates a mutex handle.

Syntax

```
public cdecl void** allocate_mutex_handle();
```

Returns

void**

Returns pointer to a new uninitialized mutex handle.

1.2.3 allocate_thread_data(ulong) Function

Allocates thread specific data from the thread table: an exception table, a call stack and a frame pointer.

Syntax

```
public cdecl void allocate_thread_data(ulong tid);
```

Parameters

Name	Type	Description
tid	ulong	Thread identifier.

1.2.4 begin_capture_call_stack() Function

Locks a mutex that provides access to a global memory for a call stack string.

Syntax

```
public cdecl void begin_capture_call_stack();
```

1.2.5 broadcast_cond(void**) Function

Unblocks all threads that wait on a condition variable.

Syntax

```
public cdecl int broadcast_cond(void** cond);
```

Parameters

Name	Type	Description
cond	void**	A pointer to a condition variable handle.

Returns

int

Returns 0 if call succeeds, or a non-zero error code otherwise.

1.2.6 capture_call_stack() Function

Formats a call stack string for the calling thread to a global call stack buffer and returns a pointer to it.

Syntax

```
public cdecl const char* capture_call_stack();
```

Returns

const char*

Returns a pointer to a call stack string for the calling thread.

Remarks

Before calling this function, call begin_capture_call_stack() function, and after the string is consumed, call end_capture_call_stack() function.

1.2.7 close(int) Function

Closes a file.

Syntax

```
public cdecl int close(int fd);
```

Parameters

Name	Type	Description
fd	int	A file descriptor of an open file.

Returns

int

Returns 0 if closing is successful, -1 if closing fails.

Remarks

The file descriptor can be obtained for example by calling the open_file(const char*, OpenFlags, int) function.

1.2.8 create_thread(ulong*, thread_fun, void*) Function

Creates a thread and starts executing the thread start function asynchronously.

Syntax

```
public cdecl int create_thread(ulong* tid, thread_fun start, void* arg);
```

Parameters

Name	Type	Description
tid	ulong*	Pointer to memory location that receives identifier of the created thread.
start	thread_fun	A thread start function delegate.
arg	void*	An argument received by the thread start function.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.9 dbgheap_done() Function

Frees memory allocated for the debug heap.

Syntax

```
public cdecl void dbgheap_done();
```

1.2.10 dbgheap_free(void*) Function

Frees a memory block allocated from the debug heap.

Syntax

```
public cdecl void dbgheap_free(void* mem);
```

Parameters

Name	Type	Description
mem	void*	Pointer to a memory block allocated from the debug heap.

1.2.11 dbgheap_get_serial(void*) Function

Returns allocation serial number for a memory block allocated from the debug heap.

Syntax

```
public cdecl int dbgheap_get_serial(void* mem);
```

Parameters

Name	Type	Description
mem	void*	Pointer to a memory block allocated from the debug heap.

Returns

int

Returns allocation serial number.

1.2.12 dbgheap_init() Function

Initializes the debug heap.

Syntax

```
public cdecl void dbgheap_init();
```

1.2.13 dbgheap_malloc(ulong, const char*, int) Function

Allocates a memory block from the debug heap.

Syntax

```
public cdecl void* dbgheap_malloc(ulong size, const char* file, int line);
```

Parameters

Name	Type	Description
size	ulong	The size of the memory block.
file	const char*	Source file name of the allocation.
line	int	Line number of the allocation.

Returns

void*

Returns a pointer to the allocated memory block.

1.2.14 dbgheap_report() Function

Prints a report of detected memory leaks.

Syntax

```
public cdecl void dbgheap_report();
```

1.2.15 dbgheap_stop(int) Function

Sets the stop serial number of an allocation. When dbgheap_malloc(ulong, const char*, int) function reaches the stop serial number, it prints the source file name and line number of that allocation.

Syntax

```
public cdecl void dbgheap_stop(int allocation);
```

Parameters

Name	Type	Description
allocation	int	Serial number of an allocation.

1.2.16 destroy_cond(void**) Function

Destroyes a condition variable.

Syntax

```
public cdecl int destroy_cond(void** cond);
```

Parameters

Name	Type	Description
cond	void**	A pointer to the handle of the condition variable to destroy.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.17 destroy_mutex(void**) Function

Destroyes a mutex.

Syntax

```
public cdecl int destroy_mutex(void** mtx);
```

Parameters

Name	Type	Description
mtx	void**	Poiter to the handle of the mutex to destroy.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.18 detach_thread(ulong) Function

Detaches a thread.

Syntax

```
public cdecl int detach_thread(ulong tid);
```

Parameters

Name	Type	Description
tid	ulong	The identifier of the thread to detach.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.19 directory_exists(const char*) Function

Returns 1 if a directory for the given path name exists, 0 otherwise.

Syntax

```
public cdecl int directory_exists(const char* directoryPath);
```

Parameters

Name	Type	Description
directoryPath	const char*	A path to a directory.

Returns

int

Returns 1 if directory exists, 0 otherwise.

1.2.20 end_capture_call_stack() Function

Unlocks a mutex that provides access to a global memory for a call stack string.

Syntax

```
public cdecl void end_capture_call_stack();
```

1.2.21 enter_frame(const char*, const char*, int) Function

Adds a call frame to the call stack of the calling thread.

Syntax

```
public cdecl void enter_frame(const char* fun, const char* file, int line);
```

Parameters

Name	Type	Description
fun	const char*	Name of a function.

file	const char*	Source file.
line	int	Source line.

1.2.22 equal_thread(ulong, ulong) Function

Returns true if the given thread identifiers refer to the same thread, false otherwise.

Syntax

```
public cdecl bool equal_thread(ulong tid1, ulong tid2);
```

Parameters

Name	Type	Description
tid1	ulong	First thread identifier.
tid2	ulong	Second thread identifier.

Returns

bool

Returns true if the given thread identifiers refer to the same thread, false otherwise.

1.2.23 exit(int) Function

Terminates the calling process after doing a cleanup.

Syntax

```
public cdecl void exit(int exitCode);
```

Parameters

Name	Type	Description
exitCode	int	An exit code to return to the system or to the calling parent process.

Remarks

Typically an exit code 0 indicates successful execution status and other values indicate that an error of some kind occurred.

1.2.24 exit_thread(void*) Function

Exits the current thread.

Syntax

```
public cdecl void exit_thread(void* value);
```

Parameters

Name	Type	Description
value	void*	Ignored.

1.2.25 file_exists(const char*) Function

Returns 1 if a file for the given path name exists, 0 otherwise.

Syntax

```
public cdecl int file_exists(const char* filePath);
```

Parameters

Name	Type	Description
filePath	const char*	A path to a file.

Returns

int

Returns 1 if file exists, 0 otherwise.

1.2.26 free(void*) Function

Returns a block of memory back to the system.

Syntax

```
public cdecl void free(void* mem);
```

Parameters

Name	Type	Description
mem	void*	A pointer to the memory block obtained by calling the malloc(ulong) function, or null.

1.2.27 free_cond_handle(void**) Function

Frees a condition variable handle.

Syntax

```
public cdecl void free_cond_handle(void** handle);
```

Parameters

Name	Type	Description
handle	void**	Pointer to a condition variable handle.

1.2.28 free_mutex_handle(void**) Function

Frees a mutex handle.

Syntax

```
public cdecl void free_mutex_handle(void** handle);
```

Parameters

Name	Type	Description
handle	void**	Pointer to a mutex handle.

1.2.29 free_thread_data(ulong) Function

Frees thread specific data from the thread table.

Syntax

```
public cdecl void free_thread_data(ulong tid);
```

Parameters

Name	Type	Description
tid	ulong	Thread identifier.

1.2.30 get_current_working_directory(char*, int) Function

Obtains a path to the current working directory for the calling process.

Syntax

```
public cdecl char* get_current_working_directory(char* buf, int bufSize);
```

Parameters

Name	Type	Description
buf	char*	A buffer for the returned string.

bufSize	int	The size of the buffer.

Returns

char*

Returns the path to the current working directory, or null if the call failed.

Remarks

In Windows the functions returns the path to the current working directory for the default drive.

1.2.31 get_default_pmode() Function

Returns the default permission mode for the platform.

Syntax

```
public cdecl int get_default_pmode();
```

Returns

int

Returns the default permission mode for the platform to use in pmode parameter of the open_file(const char*, OpenFlags, int) function.

Remarks

In Windows returns S_IREAD | S_IWRITE, in Unix returns S_IRUSR | S_IWUSR | S_IRGRP.

1.2.32 get_environment_variable(const char*) Function

Gets a value of an environment variable.

Syntax

```
public cdecl char* get_environment_variable(const char* varName);
```

Parameters

Name	Type	Description
varName	const char*	The name of the environment variable.

Returns

char*

Returns a pointer to the entry for the environment variable, or null if there is no environment variable of the given name.

Remarks

(1) In Windows the name of the environment variable is not case sensitive. (2) It is not safe to modify the value through the returned pointer.

1.2.33 get_errno() Function

Returns the value of the C run-time library errno variable.

Syntax

```
public cdecl int get_errno();
```

Returns

int

Returns the value of the C run-time library errno variable.

Remarks

If the latest library call returned -1, this function returns the error condition code of the C run-time library for the latest function call. An error message can be obtained by passing the returned value to the strerror(int) function.

1.2.34 get_exception_table_addr(ulong) Function

Returns the address of the exception table for the given thread.

Syntax

```
public cdecl void* get_exception_table_addr(ulong tid);
```

Parameters

Name	Type	Description
tid	ulong	Thread identifier.

Returns

void*

Returns the address of the exception table for the given thread.

1.2.35 init_cond(void**) Function

Initializes a condition variable.

Syntax

```
public cdecl int init_cond(void** cond);
```

Parameters

Name	Type	Description
cond	void**	A pointer to a condition variable handle to initialize.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.36 init_mutex(void**) Function

Initializes a mutex.

Syntax

```
public cdecl int init_mutex(void** mtx);
```

Parameters

Name	Type	Description
mtx	void**	A pointer to a mutex handle to initialize.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.37 init_recursive_mutex(void**) Function

Initializes a recursive mutex.

Syntax

```
public cdecl int init_recursive_mutex(void** mtx);
```

Parameters

Name	Type	Description
mtx	void**	A pointer to a recursive mutex handle to initialize.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.38 join_thread(ulong, void**) Function

Waits for the given thread to terminate.

Syntax

```
public cdecl int join_thread(ulong tid, void** value);
```

Parameters

Name	Type	Description
tid	ulong	Threads identifier.
value	void**	Ignored.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.39 leave_frame() Function

Removes a call frame to the call stack of the calling thread.

Syntax

```
public cdecl void leave_frame();
```

1.2.40 lock_mutex(void**) Function

Locks a mutex. If the mutex is currently locked, waits for the mutex to become unlocked.

Syntax

```
public cdecl int lock_mutex(void** mtx);
```

Parameters

Name	Type	Description
mtx	void**	A pointer to a mutex handle.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.41 lseek(int, long, int) Function

Moves the file pointer to the specified location.

Syntax

```
public cdecl long lseek(int fd, long offset, int origin);
```

Parameters

Name	Type	Description
fd	int	A file descriptor of an open file.
offset	long	The number of bytes from origin.
origin	int	An initial position: SEEK_SET, SEEK_CUR, or SEEK_END.

Returns

long

Returns the offset of the file position from the beginning of the file after the seek operation, or -1 if the call failed.

1.2.42 malloc(ulong) Function

Obtains a block of memory from the system.

Syntax

```
public cdecl void* malloc(ulong size);
```

Parameters

Name	Type	Description
size	ulong	The number of bytes to obtain.

Returns

void*

Returns pointer to the obtained memory block, or null if there was insufficient memory available.

Remarks

The actual number of bytes obtained may be larger than the number of bytes requested for the alignment and maintenance purposes.

1.2.43 mutextbl_done() Function

Destroyes mutexes in the mutex table and frees the memory allocated for the mutex table.

Syntax

```
public cdecl void mutextbl_done();
```

1.2.44 mutextbl_init(int) Function

Allocates memory for the mutex table for the given number of mutexes and initializes the mutexes.

Syntax

```
public cdecl void mutextbl_init(int numMutexes);
```

Parameters

Name	Type	Description
numMutexes	int	The number of mutexes.

1.2.45 mutextbl_lock_mtx(int) Function

Locks a mutex that has given number.

Syntax

```
public cdecl int mutextbl_lock_mtx(int mid);
```

Parameters

Name	Type	Description

mid	int	The number of the mutex to lock.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.46 mutextbl_unlock_mtx(int) Function

Unlocks a mutex that has given number.

Syntax

```
public cdecl int mutextbl_unlock_mtx(int mid);
```

Parameters

Name	Type	Description
mid	int	The number of the mutex to unlock.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.47 open_file(const char*, OpenFlags, int) Function

Opens a file with the specified name, mode and permissions.

Syntax

```
public cdecl int open_file(const char* filename, OpenFlags openFlags, int pmode);
```

Parameters

Name	Type	Description
filename	const char*	The name of the file to open.
openFlags	OpenFlags	An open mode.
pmode	int	A permission mode if create flag is specified, otherwise ignored. Permission mode is formed by combining the following constants with OR-operation: Windows: S_IREAD, S_IWRITE; Unix: S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH

Returns

int

Returns a file descriptor (a non-negative integer) for the opened file, or -1 if the opening fails.

1.2.48 path_exists(const char*) Function

Return 1 if a given path name exists, 0 otherwise.

Syntax

```
public cdecl int path_exists(const char* path);
```

Parameters

Name	Type	Description
path	const char*	A path to a file, directory, device, named pipe, symbolic link or socket.

Returns

int

Returns 1 if a given path exits, 0 otherwise.

1.2.49 rand() Function

Returns a pseudorandom integer.

Syntax

```
public cdecl int rand();
```

Returns

int

Returns a pseudorandom integer in the range 0 ... RAND_MAX for the system. In Windows and Linux RAND_MAX is 32767.

Remarks

The returned value is the next value in a sequence from the current seeed value. Use srand(uint) function to set the initial seed value. If no initial seed value is set, 1 is used as the seed value.

1.2.50 read(int, void*, uint) Function

Reads data from a file.

Syntax

```
public cdecl int read(int fd, void* buf, uint size);
```

Parameters

Name	Type	Description
fd	int	A file descriptor referring to an open file.
buf	void*	A memory buffer to read to.
size	uint	Maximum number of bytes to read.

Returns

int

Returns the number of bytes read, or -1 in case of an error.

Remarks

The number of bytes read might be less than than size parameter if end of file encountered, or end of line encountered if the file was opened in text mode.

1.2.51 signal_cond(void**) Function

Unblocks one thread waiting on a condition variable.

Syntax

```
public cdecl int signal_cond(void** cond);
```

Parameters

Name	Type	Description
cond	void**	A pointer to a condition variable handle.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

1.2.52 sleep(long, int) Function

Puts the calling thread to sleep for the given number of seconds and nanoseconds.

Syntax

```
public cdecl int sleep(long secs, int nanosecs);
```

Parameters

Name	Type	Description
secs	long	The number of seconds to sleep.
nanosecs	int	The number of nanoseconds to sleep.

Returns

int

Returns 0, if the call succeeds, or nonzero error code otherwise.

Remarks

sleep.long.int.nanosecs must be greater or equal to zero and less than 1000000000.

1.2.53 srand(uint) Function

Sets the seed value of the pseudorandom number generator.

Syntax

```
public cdecl void srand(uint seed);
```

Parameters

Name	Type	Description
seed	uint	A seed value.

Remarks

The pseudorandom number generator generates always the same sequence of values for the same seed value. If no initial seed value is set, 1 is used as the seed value.

1.2.54 strerror(int) Function

Gets a system error message for the given C run-time library error code (errno).

Syntax

```
public cdecl char* strerror(int errnum);
```

Parameters

Name	Type	Description
errnum	int	An error code.

Returns

char*

Returns a pointer to the error message.

Remarks

Subsequent calls by this function may overwrite the contents of the error message buffer, so the returned pointer should be used before a subsequent call.

1.2.55 system(const char*) Function

Executes a command.

Syntax

```
public cdecl int system(const char* command);
```

Parameters

Name	Type	Description
command	const char*	A command possibly with arguments to pass to the system command interpreter or shell.

Returns

int

Returns the value returned by the executed command, or -1 in case of an error.

1.2.56 this_thread() Function

Returns the identifier of the calling thread.

Syntax

```
public cdecl ulong this_thread();
```

Returns

ulong

Returns the identifier of the calling thread.

1.2.57 threadtbl_done() Function

Frees the memory allocated for the thread table.

Syntax

```
public cdecl void threadtbl_done();
```

1.2.58 threadtbl_init(int) Function

Allocates memory for the thread table.

Syntax

```
public cdecl void threadtbl_init(int numExceptions);
```

Parameters

Name	Type	Description
numExceptions	int	Sets the number of exceptions that can be thrown and catched in the current program.

1.2.59 time_nanosecs(long&, int&) Function

Gets the current time from the real time clock of the computer.

Syntax

```
public cdecl int time_nanosecs(long& secs, int& nanosecs);
```

Parameters

Name	Type	Description
secs	long&	Pointer to a memory location receiving the number of seconds elapsed from the midnight 1.1.1970.

nanosecs	int&	Pointer to a memory location receiving the nanoseconds elapsed addition to elapsed seconds.

Returns

int

Returns 0, if the call succeeds, or a nonzero error code otherwise.

1.2.60 timedwait_cond(void**, void**, long, int, bool&) Function

Waits for a condition variable to become signaled or given time to elapse.

Syntax

```
public cdecl int timedwait_cond(void** cond, void** mtx, long secs, int nanosecs,
bool& timedout);
```

Parameters

Name	Type	Description
cond	void**	A pointer to a condition variable handle.
mtx	void**	A pointer to a mutex handle.
secs	long	The number of seconds to wait.
nanosecs	int	The number of nanoseconds to wait in addition to seconds.
timedout	bool&	A pointer to a memory location that receives value 1 if the given time has elapsed without the condition variable to become signaled, 0 otherwise.

Returns

int

Returns 0, if the call succeeds, or a nonzero error code otherwise.

Remarks

Atomically unlocks the given mutex and waits and atomically again locks the mutex when the call returns.

1.2.61 trylock_mutex(void**, bool&) Function

Locks a mutex and sets a Boolean variable pointed by already_locked to 0, if the mutex is currently unlocked, otherwise sets a Boolean variable pointed by to 1.

Syntax

```
public cdecl int trylock_mutex(void** mtx, bool& already_locked);
```

Parameters

Name	Type	Description
mtx	void**	A mutex to try to lock.
already_locked	bool&	A pointer to a Boolean variable that receives 0, if the mutex is currently unlocked, or 1 if the mutex is currently locked.

Returns

int

Returns 0, if the call succeeds, or a nonzero error code otherwise.

1.2.62 unlock_mutex(void**) Function

Unlocks a mutex.

Syntax

```
public cdecl int unlock_mutex(void** mtx);
```

Parameters

Name	Type	Description
mtx	void**	A pointer to a mutex handle to unlock.

Returns

int

Returns 0, if the call succeeds, or a nonzero error code otherwise.

1.2.63 wait_cond(void**, void**) Function

Waits a condition variable to become signaled.

Syntax

```
public cdecl int wait_cond(void** cond, void** mtx);
```

Parameters

Name	Type	Description
cond	void**	A pointer to a condition variable handle.
mtx	void**	A pointer to a mutex handle.

Returns

int

Returns 0, if the call succeeds, or a nonzero error code otherwise.

Remarks

Atomically unlocks the given mutex and waits and atomically again locks the mutex when the call returns.

1.2.64 write(int, void*, uint) Function

Writes data to a file.

Syntax

```
public cdecl int write(int fd, void* buf, uint size);
```

Parameters

Name	Type	Description
fd	int	A file descriptor of an open file.
buf	void*	A buffer of data to write.
size	uint	The number of bytes to write.

Returns

int

Returns the number of bytes written, or -1 in case of an error.

1.3 Delegates

Delegate	Description
thread_fun	A thread start function delegate.

1.3.65 thread_fun Delegate

A thread start function delegate.

Syntax

```
thread_fun
```

Parameters

Name	Type	Description
arg	void*	An argument to the thread start function.

1.4 Enumerations

Enumeration	Description
OpenFlags	Specifies file open mode for open_file(const char*, OpenFlags, int) function. Open mode is formed by combining one or more enumeration constants with OR-operation.

1.4.65.1 OpenFlags Enumeration

Specifies file open mode for open_file(const char*, OpenFlags, int) function. Open mode is formed by combining one or more enumeration constants with OR-operation.

Enumeration Constants

Constant	Value	Description
none	0	No flags specified.
readOnly	1	Opens a file for reading.
writeOnly	2	Opens a file for writing.
readWrite	4	Opens a file for both reading and writing.
create	8	Creates and opens a file for writing. Has no effect is file exists unless truncate is specified also.
append	16	Opens a file for appending: moves the file pointer to the end of the file before each write operation.
truncate	32	Opens a file and truncates it to zero length. If specified with create, opens a new file or creates a new file.
text	64	Windows: Opens a file in text mode.
binary	128	Windows: Opens a file in binary mode.

1.5 Constants

Constant	Type	Value	Description
SEEK_CUR	int	1	Offset parameter of the lseek(int, long, int) function represents a position from the current file position.
SEEK_END	int	2	Offset parameter of the lseek(int, long, int) function represents a position from the end of the file onward.
SEEK_SET	int	0	Offset parameter of the lseek(int, long, int) function represents a position from the beginning of the file.
S_IREAD	int	256	Windows: Reading permitted.
S_IRGRP	int	32	Unix: Reading by group permitted.
S_IROTH	int	4	Unix: Reading by others permitted.
S_IRUSR	int	256	Unix: Reading by user permitted.
S_IWGRP	int	16	Unix: Writing by group permitted.
S_IWOTH	int	2	Unix: Writing by others permitted.
S_IWRITE	int	128	Windows: Reading and writing permitted.
S_IWUSR	int	128	Unix: Writing by user permitted.
S_IXGRP	int	8	Unix: Executing by group permitted.
S_IXOTH	int	1	Unix: Executing by others permitted.
S_IXUSR	int	64	Unix: Executing by user permitted.

stderr	int	2	Standard error file descriptor.
stdin	int	0	Standard input file descriptor.
stdout	int	1	Standard output file descriptor.