Gravel2 Kernel API

18–342 Fundamentals of Embedded Systems

November 12, 2013

1 Kernel API

This document lists all the syscalls that the Gravel real-time kernel supports. All syscalls are to be made using the SWI instruction. The SWI number to be used is the syscall number of your syscall added to a base of 0x900000. Arguments to a syscall are put in order in r0, r1, r2, r3. The return value of a syscall is put in r0. No registers other than r0 are modified during a syscall. Control is transfered to the instruction succeeding the SWI after the syscall is complete. A syscall with an invalid syscall number results in the task being forcibly exited with an exit status of 0x0badc0de. A syscall returning an error returns the negative of the error codes mentioned below.

2 SWI Numbers

SWI Number	Syscall Name
`0x900003`	`read`
`0x900004`	`write`
`0x900006`	`time`
`0x900007`	`sleep`

3 Syscalls

3.1 read

Syscall Number 3

Description `read` attempts to read a requested number of bytes from the given file descriptor. The read values are written into the provided buffer sequentially.

Argument 0 `int fd`
The file descriptor to read from — e.g. STDIN_FILENO

Argument 1 `void *buf`
The target buffer to write any read data to.

Argument 2 `size_t count`
The number of bytes to read into `buf`. If this value is zero, then zero is returned and nothing else happens. If this value is greater than SSIZE_MAX, the result is unspecified.

Return `ssize_t ret`
On success, the number of bytes read is returned. This number can be less than `count` if a short read occurs due to an end-of-file or another error. The following errors may be returned.

EBADF `fd` was an invalid or unreadable file descriptor.

EFAULT `buf` points to region whose bounds lie outside valid address space.

EIO An internal I/O error occurred.

3.2 write

Syscall Number 4

Description `write` attempts to write a requested number of bytes to the given file descriptor. The values to be written are read from the provided buffer sequentially.

Argument 0 `int fd`
The file descriptor to write to — e.g. STDOUT_FILENO

Argument 1 `const void *buf`
The target buffer to read data from.

Argument 2 `size_t count`
The number of bytes to read from `buf`. If this value is zero, then zero is returned and nothing else happens. If this value is greater than SSIZE_MAX, the result is unspecified. Note that this behavior is subtly different from the linux behavior and is used here to remain consistent with read.

Return `ssize_t ret`
On success, the number of bytes written is returned. This number can be less than `count` if the underlying medium cannot accept any more bytes at the moment. The following errors may be returned on failure.

EBADF `fd` was an invalid or unwritable file descriptor.

EFAULT `buf` points to region whose bounds lie outside valid address space.

EIO An internal I/O error occurred.

3.3 time

Syscall Number 6

Description `time` returns the time in milliseconds that have elapsed since the kernel booted up. This can be interpreted as the number of milliseconds to ellapse since the kernel timer was turned on.

Return `unsigned long ret`
The function returns the time elapsed since the kernel booted up in milliseconds. There is no error case. Note that even though the number returned is in milliseconds, the actual resolution may be lower.

3.4 sleep

Syscall Number 7

Description `sleep` suspends the execution of the current task for the given time. The task may continue executing after atleast the given time has elapsed.

Argument 0 `unsigned long millis`
The number of milliseconds to sleep. Note that the kernel may round up the time provided if its timer resolution is lower than a millisecond.

Return `void`
There is no return value. The function always succeeds.

3.5 task_create

Syscall Number 10

Description `task_create` takes a set of tasks, verifies that they are schedulable, and then begins to schedule them. All currently present tasks are cancelled, and an entirely new task set is scheduled upon success.

Argument 0 `task_t *`
An array of task descriptors that qualify the tasks to create.

Argument 0 `size_t num_tasks`
The number of elements in the array.

Return `int`
Upon success, this function does not return. It instead begins scheduling tasks. On failure, an error code may be returned.

EINVAL `num_tasks` is greater than the maximum number of tasks the OS supports (64).

EFAULT `tasks` points to region whose bounds lie outside valid address space.

ESCHED The given task set is not schedulable – some tasks may not meet their deadlines.

3.6 mutex_create

Syscall Number 15

Description This function returns a unique identifier to a mutex object that the kernel has created. If the kernel is out of identifiers, it returns an error. All valid ids are positive integers. The returned ids start at zero and monotonically increase by one for every successful subsequent call. This function should return successfully for at least 32 invocation.

Return `int`
A positive integer is returned if successful. Upon failure, an error code may be returned.

ENOMEM The kernel cannot produce any more mutices.

3.7 mutex_lock

Syscall Number 16

Description This function instructs the kernel to acquire the indicated mutex. If the mutex is acquirable, the return is zero. If the mutex is unacquirable, the kernel will put the task to sleep until is is acquirable and will then acquire it.

Argument 0 `int mutex`
The unique id for a mutex object that was obtained from a successful invocation of `mutex_create`.

Return `int`
A zero is returned upon the successful acquisition of the mutex. Upon failure, an error code may be returned.

EINVAL The provided mutex identifier is invalid.

EDEADLOCK The current task is already holding the lock.

3.8 mutex_unlock

Syscall Number 17

Description This function instructs the kernel to release an already acquired mutex. If the mutex has other tasks waiting for its release, the first waiting task will be woken up and the mutex will be handed over to that task atomically (with respect to the scheduler).

Argument 0 `int mutex`
The unique id for a mutex object that was obtained from a successful invocation of `mutex_create`.

Return `int`
A zero is returned upon the successful acquisition of the mutex. Upon failure, an error code may be returned.

EINVAL The provided mutex identifier is invalid.

EPERM The current task does not hold the mutex.

3.9 event_wait

Syscall Number 20

Description Puts the calling task to sleep on a given device numbers sleep queue until the device next signals the device event. Device events are signalled regularly based on the signalling frequency of the device.

Argument 0 `unsigned int dev`
A valid device number – 0, 1, 2 or 3.

Return `int`
A zero is returned upon the successful acquisition of the mutex. Upon failure, an error code may be returned.

EINVAL The provided device identifier is invalid.

4 Task Descriptor

```
typedef struct task
{
void (*lambda)(void*);
void *data;
void *stack_pos;
unsigned long C;
unsigned long T;
} task_t;
```

This is the definition of a task descriptor structure. `lambda` is a pointer to the function to be invoked when the task first starts. `data` is the argument that is blindly passed to `lambda`. `stack pos` is the initial value of userspace `sp` when the task first starts executing. There is no restriction on this value because it is possible to write user applications that never use the stack and hence use sp as a general purpose register. `C` and `T` are the worst case compute time and the periodicity of the task respectively. They are both measured in milliseconds. The system is allowed to round/approximate the given values based on the internal resolution of the OS timer as long as it does not incorrectly cause a task to miss its deadline. In general, this means rounding up `C` and rounding down `T`.