

Threads

Version 0.1.4 9/27/2005

Table of Contents

| | |
|--|---|
| Revision History | 2 |
| 1 The Location Where Thread Information Is Stored..... | 2 |
| 2 OSThreadInfo Thread System Information | 5 |
| 3 The OSThread Thread Structure | 5 |

Revision History

9/27/2005 Added `alarmForSleep` to `OSThread` structure descriptions.

1 The Location Where Thread Information Is Stored

Thread information (`OSThreadInfo`) is allocated in the main memory. The region storing the address region where this information is stored is in the System Work Area, which is part of main memory—it can be accessed by both ARM9/7. The System Work Area start address is:

$$\text{HW_MAIN_MEM_SYSTEM} = \text{HW_MAIN_MEM} + 0x7FFC80 = 0x27FFC80$$

This address can be acquired from within user programs using `OS_GetSystemWork()`.

In the case of ARM9, the pointer is stored to:

$$\text{HW_THREADINFO_MAIN} = \text{HW_MAIN_MEM} + 0x007FFFA0 = 0x27FFFA0$$

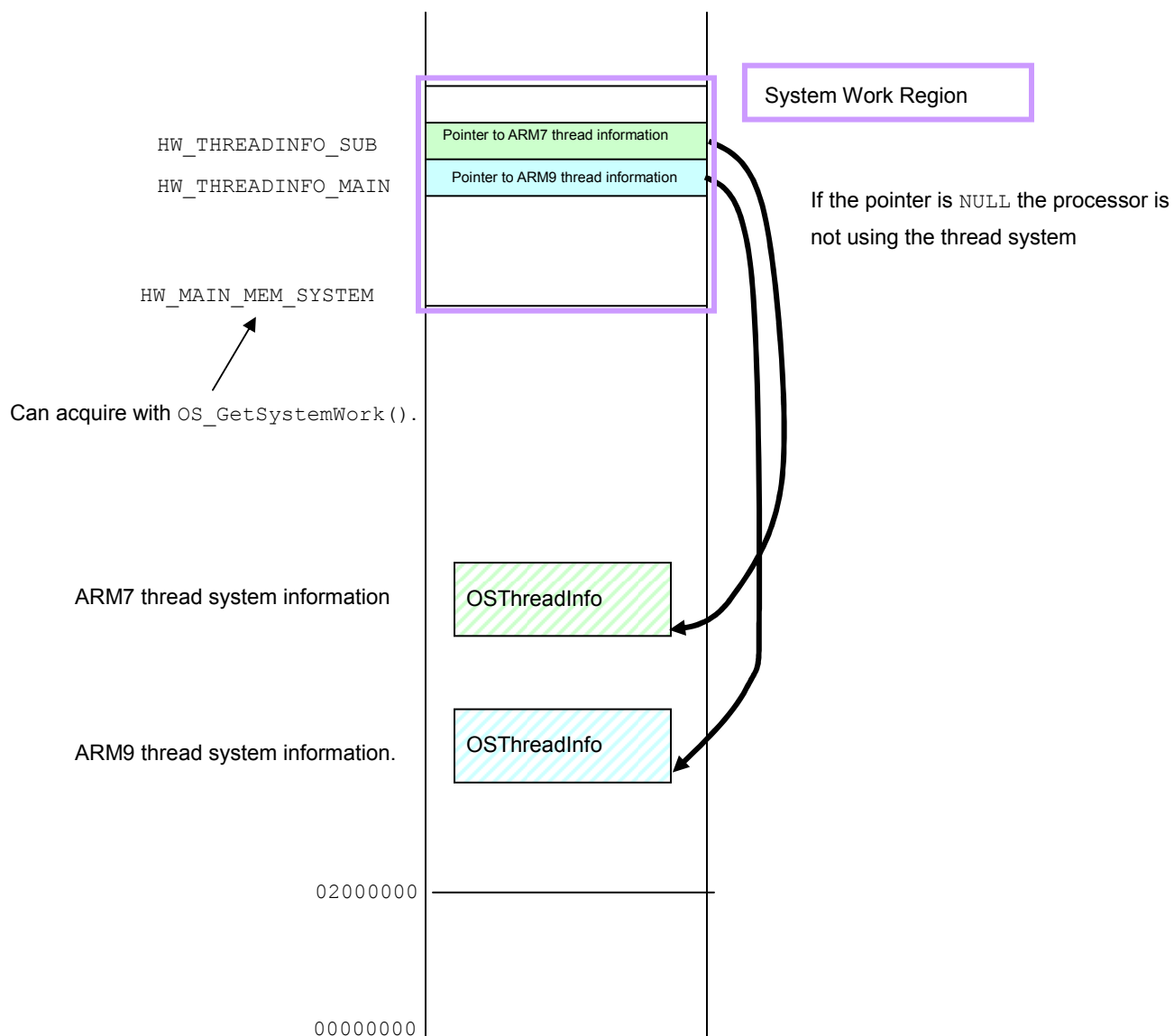
You can acquire the store address of this pointer as `OS_GetSystemWork()->threadinfo_mainp`.

In the case of ARM7, the pointer is stored to:

$$\text{HW_THREADINFO_SUB} = \text{HW_MAIN_MEM} + 0x007FFFA4 = 0x27FFFA4.$$

You can acquire the store address of this pointer as `OS_GetSystemWork()->threadinfo_subp`.

If the pointer is `NULL`, the processor is not using the thread system.



2 OSThreadInfo Thread System Information

```
// ----- Thread & context packed structure
typedef struct OSThreadInfo
{
    u16          isNeedRescheduling;
    u16          irqDepth;
    OSThread*    current;
    OSThread*    list;
    void*        switchCallback;
} OSThreadInfo;
```

Following is a description of each member of the `OSThreadInfo` structure.

- `isNeedRescheduling` is a flag for remembering whether it is necessary to reschedule when a thread switch request is generated at the time of an IRQ interrupt., and the IRQ interrupt is terminated. This flag has two values: TRUE and FALSE. Since this value is used by the OS, do not touch it.
- `irqDepth` stores the IRQ interrupt level. Since this variable is accessed by multiple interrupts and is used internally by the OS, making manual changes is strongly discouraged.
- `current` is a pointer to the thread information of the current thread.
- `list` is a pointer to the thread list. Threads are connected in order from the one having the highest priority, using the next member in `OSThread`. At the end, `next = NULL`. If no threads are registered, the list will be `NULL`.
- `switchCallback` stores the callback value during thread switching; `NULL` if no callback has been set.

3 The OSThread Thread Structure

```
// ----- Thread structure
typedef struct _OSThread OSThread;
struct _OSThread
{
    OSContext      context;
    OSThreadState  state;
    OSThread*      next;
    u32            id;
    u32            priority;
    void*          profiler;
    OSThreadQueue* queue;
    OSThreadLink   link
```

```

OSMutex*      mutex;
OSMutexQueue  mutexQueue;

u32           stackTop;      // for stack overflow
u32           stackBottom;  // for stack underflow
u32           stackWarningOffset;
OSThreadQueue  joinQueue;
void*         specific[OS_THREAD_SPECIFIC_MAX];
OSAlarm*      alarmForSleep;
OSThreadDestructor destructor;
void*         userParameter;
int           systemError;
};

```

Following is a description of each member of the `OSThread` structure.

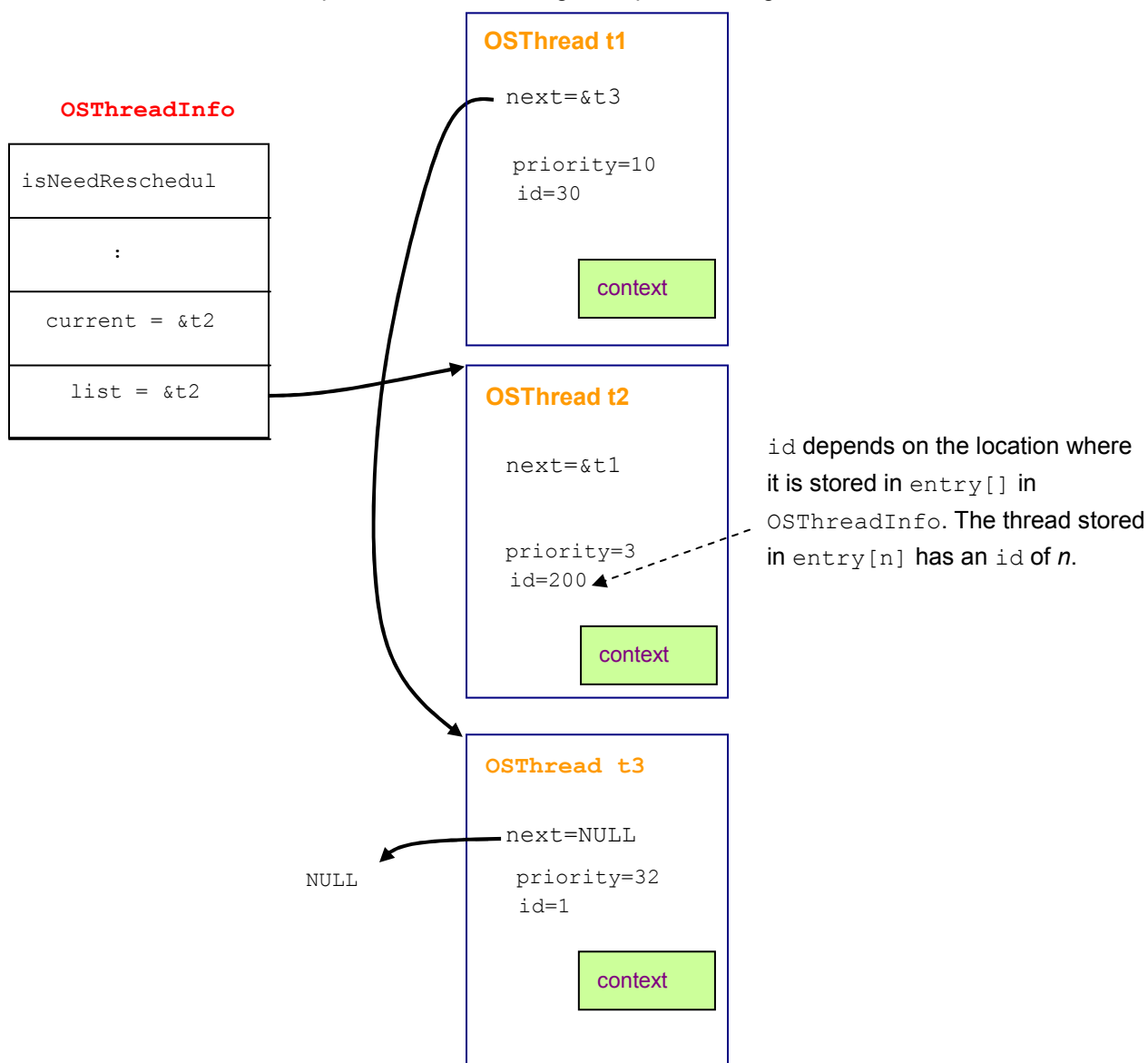
- `context` is the location at which context is stored during the time that threads are being switched.
- `state` indicates thread status:
 - `OS_THREAD_STATE_WAITING` (=0) indicates that a thread is stopped.
 - `OS_THREAD_STATE_READY` (=1) indicates that the thread is ready to run.

For a thread that has ended, `state` is `OS_THREAD_STATE_TERMINATED`.
- `next` is a pointer to the next thread when constructing a thread list. It will be `NULL` at the end.
- `id` indicates thread id. Its values are 0 – 0x7fffffff. The value is increased each time a thread is created.
- `priority` indicates the priority level of a thread. Values are 0–31. 0 indicates the thread that has the highest priority. The thread list is ordered by this thread priority. The idle thread created by `OS_InitThread()` is assigned a priority value of 32. The priority of the idle thread cannot be altered.
- **profile** is a pointer used by the profile function routines (e.g. function call tracing and function cost measurement) to store thread information. When the profile function is not used, it does nothing.
- **queue** and **link** are areas for the thread queue. `queue` stores a pointer to the thread queue specified when a thread is sleeping; `link` is link information for linking sleeping threads to the same thread queue.
- `mutex` and `mutexQueue` are parameters used for the automatic execution of the `mutex` de-allocation when the thread ends. Since the OS uses these values internally, please do not touch them.
- `stackTop`, `stackBottom`, `stackWarningOffset` are parameters used in the stack leak check. Since the OS uses these values internally, please do not touch them. They may be referenced.
- `JoinQueue`, a queue that is used to resume threads that have been sleeping when the current thread stops.
- `specific` is used internally by the system.
- `alarmForSleep` is a pointer to the alarm used when a thread sleeps.

- `destructor` is a thread destructor. It specifies the function called when the thread ends.
- `userParameter` is the user parameter. The user can use this area freely. It is neither changed nor referenced by the system.
- `systemError` is the system error value. It is used internally by the system.

Thread information example

Threads `t1`, `t2`, and `t3` are present in the following example, `t2` being the current thread.



For ARM9, the thread that is idle (priority of 32) should be the last in the list. (Although `t3` is used here, the `OSThread` structure is `OSi_IdleThread` in `os_thread.c`.) ARM7 does not have idle threads.

Thread example

In the following example, threads t1, t2, and t4 are linked to thread queue tq.

