MiniOS - API

Ein Minibetriebssystem für Mikrocontroller

User-API, Anhang zur MiniOS-Dokumentation

Version 31. März 2006, Hj. Kern, Winterthur

Inhalt

User-API

Hinweise und Legende zu Makros und Funktionen

OS-Makros und Funktionen sind *immer* Unterfunktionen von OS_Execute().

Legende

void Funktion (void)	os_userapi.h	**ISR**	Sched
API Funktion, Makro	Modul	Fn ist in ISR oder Task aufrufbar	Scheduler-Aufruf muss der Funktion folgen (*)

(*) z.B. SCHEDULE()

Listing - Beide Varianten (A) und (B) bewirken dasselbe. Im Makro (A) ist der Scheduler-Aufruf schon enthalten.

(A) Makro...

```
...
WAIT(100)
...
```

(B) Funktions- und nachfolgender Makroaufruf...

```
...
OS_WaitTimer(100);    // Timer-Funktion
SCHEDULE()            // Scheduler-Makro
...
```

API Datentypen

Typ		*für*
typedef unsigned char	BOOL	Boolsche Werte, allgemeiner Gebrauch Dazu sind TRUE = 1 und FALSE = 0 definiert.
typedef unsigned char	UBYTE	Byte, allgemeiner Gebrauch
typedef unsigned int	T_TICK	Systemzeit, Tick-Zähler
typedef unsigned char	T_TIMER	Task-Delay, Task-SW-Timer

Falls ein erweiterter Bereich des Tick-Zählers oder Task-Timers gewünscht wird, müssen die entsprechenden Typen angepasst werden.

Beispiel:

```
typedef  unsigned long  T_TICK
```

API Makros

STATE	OS-Makro	os_userapi.h

System-Variable

OS.SysState, sie zeigt, in welchem Zustand sich das System gerade befindet.

OS_STOPPED	OS gestoppt
OS_RUN_SYSTEM	OS im Scheduler
OS_RUN_IDLE	OS im Idle-Modus
OS_RUN_USER	OS in einer User-Task

CURRENT	OS-Makro	os_userapi.h

System-Variable

OS.CurrentTask, ID-Nummer der aktuell rechnenden Task.

ERROR	OS-Makro	os_userapi.h

System-Variable

OS.ExceptionCode, OS-Fehlercode

NOTIFY	OS-Makro	os_userapi.h

System-Variable

OS.NotifyCount, OS Ereignisregistrierung, Zähler für eingetroffene OS-Ereignisse

CLR_NOTIFY()	OS-Makro	os_userapi.h

System-Ereignisregistrierung zurücksetzen und System-Timeout unterdrücken, indem der Ereigniszähler OS.NotifyCount = 0 gesetzt wird.

TICK	OS-Makro	os_userapi.h

System-Variable

System-Zeit in Ticks, TICK wird in der Timer-ISR aktualisiert.

CLR_TICK()	OS-Makro	os_userapi.h

System-Tick-Zähler zurücksetzen auf Wert = Null.

TIMER	OS-Makro	os_userapi.h

System-Variable

Peek Task[OS.CurrentTask].Timer, der einer Task zugeordnete SW-Timer.

CLR_TIMER()	OS-Makro	os_userapi.h

Task-Timer der gerade rechnenden Task zurücksetzen auf Wert = Null.

ON_TIMEOUT	OS-Makro	os_userapi.h

Ausdruck (OS.JumpMessage == OSJ_ON_TMR), Auswertung eines Task-Timeout-Events, Beispiel:

```
...
OS_WaitSem(SEM_A, 10);
SCHEDULE()
if (ON_TIMEOUT) {
   Timeout_Behandlung();
}
...
```

SCHEDULE ( )	OS-Makro	os_userapi.h

Task-Kontextwechsel, ruft den Scheduler unbedingt auf.

- Task-Status TS_READY wird zugewiesen
- Task weiterführen mittels Round-Robin-Scheduling als "Hintergrund-Task"

SCHED_PRIO ( )	OS-Makro	os_userapi.h

Task-Kontextwechsel, ruft den Scheduler unbedingt auf.

- Task-Status TS_READY_HIGH wird zugewiesen
- Task weiterführen mittels Prioritäts-Scheduling in der Ebene "Ereignisbehandlung"

STOP_START (tid)	OS-Makro	os_userapi.h

- Stoppt die aktuelle Task und startet die Task tid
- Die aktuelle Task erhält den Status TS_STOP
- Die Task tid erhält den Status TS_READY_HIGH, → Prio-Scheduling
- Scheduler aufrufen, um einen Task-Kontextwechsel auszuführen

UBYTE tid Task-Index der angesprochenen Task

STOP ( )	OS-Makro	os_userapi.h

- Die aktuelle Task erhält den Status TS_STOP
- Scheduler aufrufen, um einen Task-Kontextwechsel auszuführen

WAIT (Ticks)	OS-Makro	os_userapi.h

- Task-Delay starten und warten auf Timer-Ablauf
- Scheduler aufrufen, um einen Task-Kontextwechsel auszuführen

T_TIMER Ticks zu wartende Anzahl Timer-Ticks

WAIT_SEM (SemID)	OS-Makro	os_userapi.h

- Warten bis der Semaphor SemID signalisiert wird
- Scheduler aufrufen, um einen Task-Kontextwechsel auszuführen

UBYTE SemID angesprochener Semaphor (-Index)

WAIT_MSG (Msg)	OS-Makro	os_userapi.h

- Warten bis eine bestimmte Message in derMailbox eintrifft, Message-Byte wird in der Mailbox gelöscht.
- Scheduler aufrufen, um einen Task-Kontextwechsel auszuführen

UBYTE Msg erwartete und abzuholende (zu löschende) Message

API Funktionen OS-Steuerung

void OS_Execute (void)	os_kernel.h

OS und Applikation ausführen. Ruft nach dem Initialisieren des OS und der Tasks den Scheduler auf, der dann die Anwender-Tasks endlos ausführt. Sie wird innerhalb des Hauptprogramms main() platziert.

void OS_Exception (UBYTE Code)	os_userapi.h

Fehlerbehandlung, Trap: Die Funktion beendet das OS und die Applikation beim Eintreffen einer Exception und weist der globalen Error-Variablen einen Fehler-Code zu. Siehe Abschnitt Fehlerbehandlung.

UBYTE Code	aktuellen Fehlercode der globalen Error-Variable zuweisen

void OS_ClearException (void)	os_userapi.h

Fehlerstatus des OS auf "keine Fehler" zurücksetzen, alle Systemfehlervariablen löschen.

void OS_NotifyEvent (void)	os_userapi.h

Ereignis-Registrierung innerhalb einer Task. Der Ereigniszähler wird bei jedem eintreffenden Ereignis innerhalb einer Task erhöht. Gleichzeitig wird überwacht, dass nicht zu viele (>= OS_NOTIFY_MAX) unbearbeitete Ereignisse anstehen. Jeder Scheduler-Eintritt setzt den Zähler zurück.

ERROR: ERR_EVENT_NOTIFY	OS beenden, falls die Grenze der Anzahl unbearbeiteter Ereignisse OS_NOTIFY_MAX überschritten wird.

void OS_Notify_ISR (void)	os_userapi.h	**ISR (*)**

(*) nur in ISR anwenden

Registrierung einer ISR. Der Ereigniszähler wird in jeder ISR welche diese Funktion aufruft erhöht. Gleichzeitig wird überwacht, dass nicht zu viele (>= OS_NOTIFY_MAX) unbearbeitete ISR-Ereignisse anstehen. Jeder Scheduler-Eintritt setzt den Zähler zurück.

Wird auch von den Funktionen OS_SignalSem_ISR() und OS_TimerTick() aufgerufen.

ERROR: ERR_ISR_NOTIFY	OS beenden, falls die Grenze der Anzahl unbearbeiteter ISR-Ereignisse OS_NOTIFY_MAX überschritten wird.

void OS_ClearNotifier (void)	os_userapi.h	**ISR**

System-Timeout unterdrücken, indem der Ereigniszähler auf den Zählwert Null zurückgesetzt wird.

UBYTE OS_PeekNotifier (void)	os_userapi.h	**ISR**

Aktuellen Zählwert des Ereigniszählers zurückgeben, nicht blockierend.

void OS_Terminate (void);	os_userapi.h	**ISR**

OS und Applikation sofort beenden.

API Funktionen Task-Steuerung

void OS_StopCurrentTask (void)	os_userapi.h	Sched

Die aktuelle Task erhält den Status TS_STOP und wird vom Scheduler nicht mehr weiter ausgeführt.

void OS_StopTask (UBYTE tid)	os_userapi.h	Sched

Die Task tid erhält den Status TS_STOP und wird vom Scheduler nicht mehr weiter ausgeführt.

UBYTE tid	Task-Index der angesprochenen Task
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

void OS_StartTask (UBYTE tid)	os_userapi.h	Sched

Die Task tid erhält den Zustand TS_READY_HIGH zugewiesen, damit sie mittels Prioritäts-Scheduling in der Scheduler-Ebene "Ereignisbehandlung" weiterbehandelt wird.

UBYTE tid	Task-Index der angesprochenen Task
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

void OS_StartBackgroundTask (UBYTE tid)	os_userapi.h	Sched

Die Task tid erhält den Zustand TS_READY zugewiesen, damit sie mittels Round-Robin-Scheduling als "Hintergrund-Task" weiterbehandelt wird.

UBYTE tid	Task-Index der angesprochenen Task
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

API Funktionen Timer

void OS_TimerTick (void)	os_userapi.h	**ISR (*)**

(*) nur in ISR anwenden

System-Timing-Funktion: Sie muss *einmal* innerhalb einer periodisch ausgeführten Timer-ISR platziert sein und generiert die Systemzeit, als Ticks.

void OS_ClearTick (void)	os_userapi.h	**ISR**

System-Tick-Zähler zurücksetzen auf den Zählwert Null.

void OS_SetTick (T_TICK Ticks)	os_userapi.h	**ISR**

System-Tick-Zähler laden mit dem Zählwert Ticks.

T_TICK Ticks	Vorgabe Tick-Zählwert

T_TICK OS_PeekTick (void)	os_userapi.h	**ISR**

Aktuellen Zählwert des System-Tick-Zählers zurückgeben, nicht blockierend.

Rückgabe: T_TICK	Aktuelle Anzahl System-Ticks seit der letzten Rücksetzung des Tick-Zählers.

void OS_ClearTimer (void)	os_userapi.h

Task-Timer der gerade rechnenden Task zurücksetzen auf den Zählwert Null.

void OS_SetTimer (T_TIMER Ticks)	os_userapi.h

Task-Timer der gerade rechnenden Task mit dem Zählwert Ticks laden und Timer starten ("Weckfunktion"). Task wartet nicht.

Der Datentyp T_TIMER erlaubt je nach System-Ressourcen und Anwendung verschiedene maximale Timer-Laufzeiten, z.B. unsigned char, int, long etc., siehe dazu os_kernel.h für die Konfiguration.

Mögliche Abweichung des Zeitwerts "Ticks":

Tickwert **+0/-1**,

dies ist abhängig davon, ob der Interrupt gerade eben verarbeitet wurde oder noch bevorsteht.

T_TIMER Ticks zu wartende Anzahl Timer-Ticks

void OS_WaitTimer (T_TIMER Ticks)	os_userapi.h	Sched

Task-Timer der gerade rechnenden Task mit dem Zählwert Ticks laden und Timer starten. Task blockiert und wartet auf den Timer-Ablauf.

Der Datentyp T_TIMER erlaubt je nach System-Ressourcen und Anwendung verschiedene maximale Timer-Laufzeiten, z.B. unsigned char, int, long etc., siehe dazu os_kernel.h für die Konfiguration.

Mögliche Abweichung des Zeitwerts "Ticks":

Tickwert **+0/-1**,

dies ist abhängig davon, ob der Interrupt gerade eben verarbeitet wurde oder noch bevorsteht.

T_TIMER Ticks zu wartende Anzahl Timer-Ticks

T_TIMER OS_PeekTimer (void)	os_userapi.h

Verbleibende Anzahl Ticks des Timers der gerade rechnenden Task zurückgeben, nicht blockierend.

Rückgabe: T_TIMER Zählstand des Task-Timers

void OS_Halt (T_TIMER Ticks)	os_userapi.h

Blockiert die Anwendung während der Anzahl Ticks. Kein Task-Switching, System-Timeout-Exception wird unterdrückt.

Diese Funktion kann auch ausserhalb von OS_Execute() aufgerufen werden, sofern der System-Timer-Interrupt aktiviert ist.

T_TIMER Ticks zu wartende Anzahl Timer-Ticks

API Funktionen Semaphore

void OS_InitSem (UBYTE SemID, UBYTE n)	os_userapi.h	**ISR**

Semaphor initialisieren mit dem Wert n. Zum Zeitpunkt des Initialisierens darf keine Task auf den Semaphor warten. Wert n wird begrenzt auf n = (MAX_SEM_COUNT - 1). Dies ist die vorkonfigurierte Ueberlaufgrenze des zählenden Semaphors.

UBYTE SemID	angesprochener Semaphor (-Index)
UBYTE n	Initialwert des Semaphors
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

void OS_ClearSem (UBYTE SemID)	os_userapi.h	**ISR**

Semaphor initialisieren mit dem Wert Null.

UBYTE SemID	angesprochener Semaphor (-Index)
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

void OS_ClearAllSemaphores (void)	os_userapi.h

Alle Semaphore initialisieren mit dem Wert Null.

UBYTE OS_TestSemOvfl (UBYTE SemID)	os_userapi.h	**ISR**

Ermittelt die mögliche Anzahl Zählungen vor dem Ueberlauf des Semaphors, nicht blockierend.

Rückgabe = (MAX_SEM_COUNT - 1) - Zählstand des Semaphors

(Hinweis: Der Semaphor kann bis zur Zählgrenze (MAX_SEM_COUNT - 1) zählen, Werte darüber erzeugen eine Overflow-Exception.)

UBYTE SemID	angesprochener Semaphor (-Index)
Rückgabe: UBYTE	0 = Semaphor-Ueberlauf steht bevor n = Anzahl Zählungen bis zum Ueberlauf
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

void OS_SignalSem (UBYTE SemID)	os_userapi.h

Semaphor innerhalb einer Task signalisieren, d.h. Zählstand inkrementieren.

(Diese Funktion *nicht für ISR* verwenden, ISR-Version siehe unten!)

UBYTE SemID	angesprochener Semaphor (-Index)
ERROR: ERR_SEM_OVFL	OS beenden, falls die Zählgrenze MAX_SEM_COUNT des Semaphors erreicht wird.
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

void OS_SignalSem_ISR (UBYTE SemID)	os_userapi.h	**ISR (*)**

(*) nur in ISR anwenden.

Semaphor innerhalb einer ISR signalisieren, d.h. Zählstand inkrementieren.

UBYTE SemID	angesprochener Semaphor (-Index)
ERROR: ERR_ISR_SEMOVFL	OS beenden, falls die Zählgrenze MAX_SEM_COUNT des Semaphors erreicht wird.
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

void OS_WaitSem (UBYTE SemID, T_TIMER Ticks)	os_userapi.h	Sched

Warten bis der Semaphor SemID signalisiert, d.h. Zählstand grösser Null wird.

UBYTE SemID	angesprochener Semaphor (-Index)
T_TIMER Ticks	Timeout in Ticks, Null: warten ohne Timeout
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

Beispiel für Timeout-Behandlung, mittels Abfrage des Makros ON_TIMEOUT

```
if (ON_TIMEOUT) {
   // Timeout behandeln
}
else {
   // Kein Timeout
}
```

UBYTE OS_PeekSem (UBYTE SemID)	os_userapi.h	**ISR**

Semaphor prüfen ohne ihn zu verändern, nicht blockierend.

Parameter, Fehler

UBYTE SemID	angesprochener Semaphor (-Index)
Rückgabe: UBYTE	aktueller Zählstand des Semaphors
ERROR: ERR_ARRAY_BOUNDARY	OS beenden, falls die Grenze des beteiligten Arrays überschritten wird.

API Funktionen Mailbox, Messages

UBYTE OS_ClearMessage (UBYTE Msg)	os_userapi.h

Eine bestimmte Byte-Message in der Mailbox löschen. Alle Messages des selben Typs werden gelöscht. Nicht blockierend.

UBYTE Msg	nach dieser Message suchen und sie löschen
Rückgabe: UBYTE	n = gefundene Anzahl der gesuchten und gelöschten Message-Bytes 0 = gesuchte Message nicht vorhanden oder Mailbox leer

void OS_ClearMessageBuffer (void)	os_userapi.h

Alle Message-Bytes in der Mailbox löschen.

UBYTE OS_GetMessageCount (void)	os_userapi.h

Gibt die Anzahl aller vorhandenen Messages in der Mailbox zurück.

Rückgabe: UBYTE	Anzahl Message-Bytes im Mailbox-Puffer

UBYTE OS_GetMailboxSpace (void)	os_userapi.h

Testet die Mailbox auf vorhandenen Platz.

Rückgabe: UBYTE	0 = Mailbox/Message-Buffer ist voll n = n Byte-Messages finden noch Platz

void OS_PostMessage (UBYTE Msg)	os_userapi.h

Eine Byte-Message in die Mailbox speichern. Falls die Box voll ist, wird OS_Exception() aufgerufen und zur Fehlerbehandlung verzweigt.

UBYTE Msg	diese Message in der Mailbox speichern
ERROR: ERR_MSGBUF_OVFL	OS beenden, falls die Grenze der Mailbox-Kapazität MAX_MSGBUF überschritten wird.

void OS_WaitMessage (UBYTE Msg, T_TIMER Ticks)	os_userapi.h	Sched

Warten bis eine bestimmte Message in derMailbox eintrifft. Die erwartete Message-Byte wird in der Mailbox gelöscht.

UBYTE Msg	erwartete und abzuholende (zu löschende) Message
T_TIMER Ticks	Timeout in Ticks, Null: warten ohne Timeout

Beispiel für Timeout-Behandlung, mittels Abfrage des Makros ON_TIMEOUT

```
if (ON_TIMEOUT) {
   // Timeout behandeln
}
else {
   // Kein Timeout
}
```

UBYTE OS_GetMessage (UBYTE Msg)	os_userapi.h

Eine bestimmte Byte-Message in der Mailbox abholen und löschen, nicht blockierend. Falls die gesuchte Message nicht vorhanden ist wird NULL zurückgegeben.

UBYTE Msg	nach dieser Message suchen und sie abholen
Rückgabe: UBYTE	gesuchtes Message-Byte oder Null

UBYTE OS_PeekMessage (UBYTE Msg)	os_userapi.h

Nach einer bestimmten Byte-Message in der Mailbox suchen ohne sie zu löschen, nicht blockierend.

UBYTE Msg	nach dieser Message suchen
Rückgabe: UBYTE	n = gefundene Anzahl der gesuchten Message-Bytes 0 = gesuchte Message nicht vorhanden oder Mailbox leer