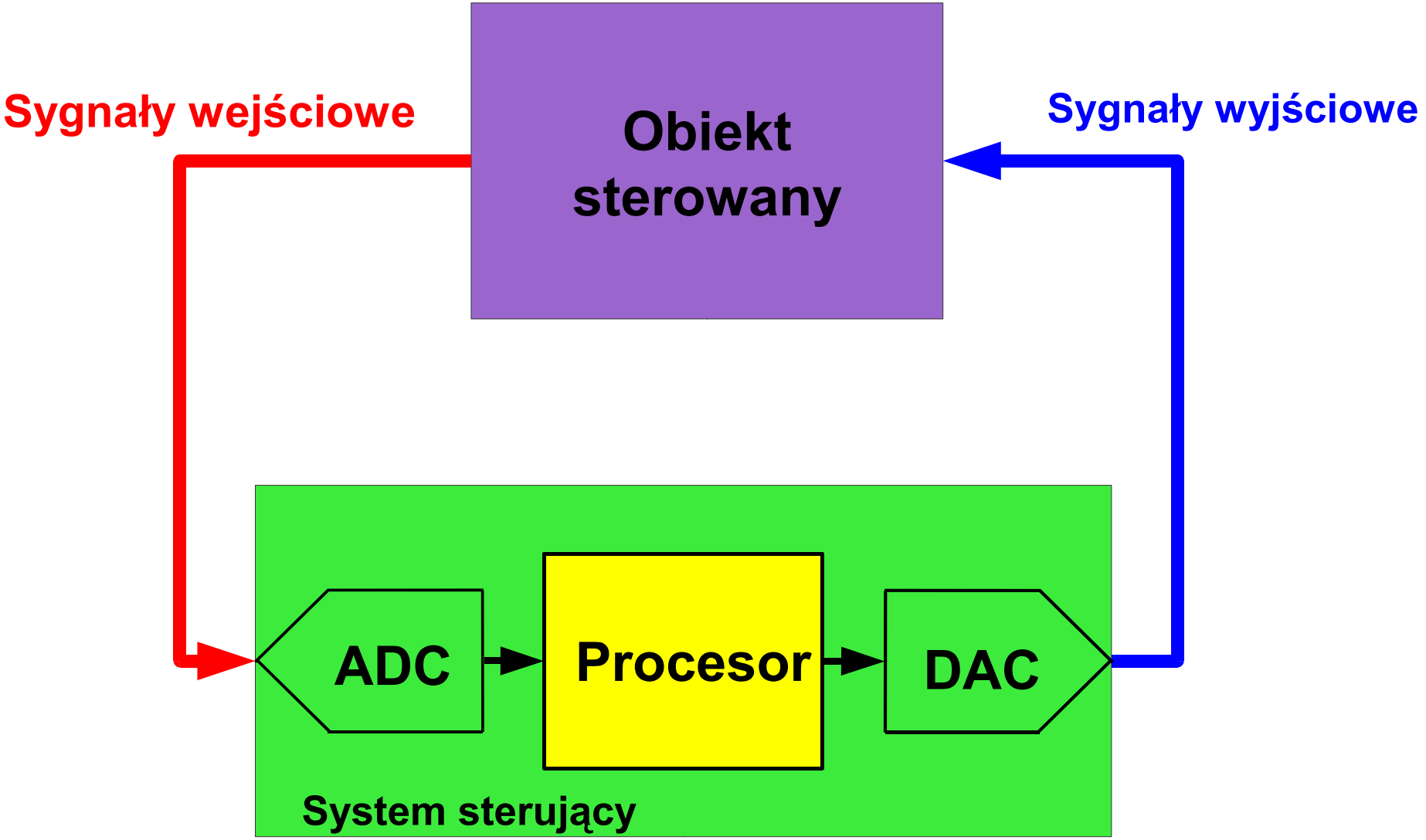
Systemy czasu rzeczywistego

System sterujący



System operacyjny

System Operacyjny OS (Operating System) - oprogramowanie, które zarządza sprzętem oraz aplikacjami komputera (procesora). Podstawą wszystkich systemów operacyjnych jest wykonywanie podstawowych zadań takich jak: zarządzanie pamięcią, przydział czasu procesora, obsługa urządzeń, ustalanie połączeń sieciowych oraz zarządzanie plikami.

Możemy wyróżnić trzy główne elementy systemu operacyjnego:

- jądro systemu wykonujące ww. zadania,
- powłoka - specjalny program komunikujący użytkownika z systemem operacyjnym,
- system plików - sposób zapisu struktury danych na nośniku.

System operacyjny czasu rzeczywistego

- Systemem czasu rzeczywistego określa się taki system, którego wynik przetwarzania zależy nie tylko od jego logicznej poprawności, ale również od czasu, w jakim został osiągnięty
- System czasu rzeczywistego odpowiada w sposób przewidywalny na bodźce zewnętrzne napływające w sposób nieprzewidywalny
- Poprawność pracy systemu czasu rzeczywistego zależy zarówno od wygenerowanych sygnałów wyjściowych jak i spełnionych zależności czasowych
- Z pojęciem „czasu rzeczywistego” wiąże się wiele nadużyć. Terminu tego używa się potocznie dla określenia obliczeń **wykonywanych bardzo szybko**, co nie zawsze jest prawdą.

System czasu rzeczywistego

- **Czas reakcji systemu** – przedział czasu potrzebny systemowi operacyjnemu na wypracowanie decyzji (sygnału wyjściowego) w odpowiedzi na zewnętrzny bodziec (sygnał wejściowy)
- **Czas reakcji systemu** może wahać się w granicach od ułamków sekund (np.: system akwizycji danych z kamery) do kilkudziesięciu godzin (np.: system sterowania poziomem wody w zbiorniku retencyjnym)
- Charakterystyka różnych zadań aplikacji musi być znana *a priori*
- Systemy czasu rzeczywistego, w szczególności systemy dynamiczne, muszą być szybkie, przewidywalne, niezawodne i adoptowalne

Podział systemów czasu rzeczywistego

- **Systemy o ostrych ograniczeniach czasowych**
  (ang. *hard real-time*) – przekroczenie terminu powoduje katastrofalne skutki (zagrożenie życia lub zdrowia ludzi, uszkodzenie lub zniszczenie urządzenia). Nie jest istotna wielkość przekroczenia terminu, a jedynie sam fakt jego przekroczenia
- **Systemy o miękkich lub łagodnych ograniczeniach czasowych** (ang. *soft real-time*) - gdy przekroczenie terminu powoduje negatywne skutki. Skutki są tym poważniejsze, im bardziej termin został przekroczony
- **Systemy o mocnych ograniczeniach czasowych**
  (ang. *firm real-time*) - gdy fakt przekroczenia terminu powoduje całkowitą nieprzydatność wypracowanego przez system wyniku. Fakt niespełnienia wymagań czasowych nie stanowi jednak zagrożenia dla ludzi lub urządzenia (bazy danych czasu rzeczywistego)

Dlaczego Linux nie jest systemem czasu rzeczywistego

- Zastosowany algorytm szeregowania z podziałem czasu
- Niska rozdzielczość zegara systemowego
- Nie wywłaszczalne jądro (nie dotyczy wersji > 2.6)
- Wyłączanie obsługi przerwań w sekcjach krytycznych
- Zastosowanie pamięci wirtualnej
- Optymalizacja wykorzystania zasobów sprzętowych

RTEMS

System operacyjny czasu rzeczywistego

Literatura

Literatura:

- Materiały wykładowe i laboratoryjne C/C++ Manuals
- Getting Started with RTEMS
- RTEMS Applications C User's Guide
- RTEMS Network Supplement
- RTEMS Shell

Literatura uzupełniająca:

- Miscellaneous Manuals
- RTEMS BSP and Device Driver Development Guide
- RTEMS CPU Supplement
- RTEMS Development Environment Guide
- RTEMS Porting Guide
- RTEMS POSIX 1003.1 Compliance Guide
- RTEMS Filesystem Design Guide

Wprowadzenie (1)

RTEMS

Jest prostym systemem operacyjnym, systemem wykonawczym czasu rzeczywistego (ang. Real Time Executive) zaprojektowanym specjalnie dla urządzeń wbudowanych. Jest to system darmowy udostępniany na zasadach licencji GNU (General Public License). System został opracowany i jest rozwijany przez OAR Corporation.

RTEMS dostarcza środowisko projektowe (ang. Development Environment):

- Kompilatory,
- Debugery,
- Wsparcie dla docelowych procesorów, tzw. BSP (ang. Board Support Package).

Wprowadzenie (2)

RTEMS
Real-Time Executive for Multiprocessor Systems

(Real Time Executive for Missile Systems)

- System operacyjny czasu rzeczywistego RTOS (Real Time Operating System) rozwijany jako projekt Open Source na licencji GPL.
- RTEMS został opracowany jako wydajny system operacyjny dla urządzeń wbudowanych.
- Dostępne są implementacje RTEMS, tzw. BSP (Board Support Packages), dla wielu procesorów: ARM, ColdFire, MC68000, Intel i960, Intel i386, MIPS, LEON, itd...

Wprowadzenie (3)

RTEMS został opracowany na potrzeby armii amerykańskiej w 1988 roku.

- Real-Time Executive for Missile Systems,
- Real-Time Executive for Military Systems,
- Real-Time Executive for Multiprocessor Systems.

- System jest nadal rozwijany i utrzymywany przez przez organizację OnLine Applications Research (OAR),
- Wsparcie dla języka C/C++ oraz Ada,
- Dostępne API:
  - Natywne RTEMS,
  - Zgodne ze standardem POSIX 1003.1b,
  - Zgodne ze standardem ITRON.

RTEMS – zastosowanie (1)

Rejestrator parametrów lotu, tzw. „czarna skrzynka”

System sterowania akceleratorem FEL

Samobieżna „kosiarka”

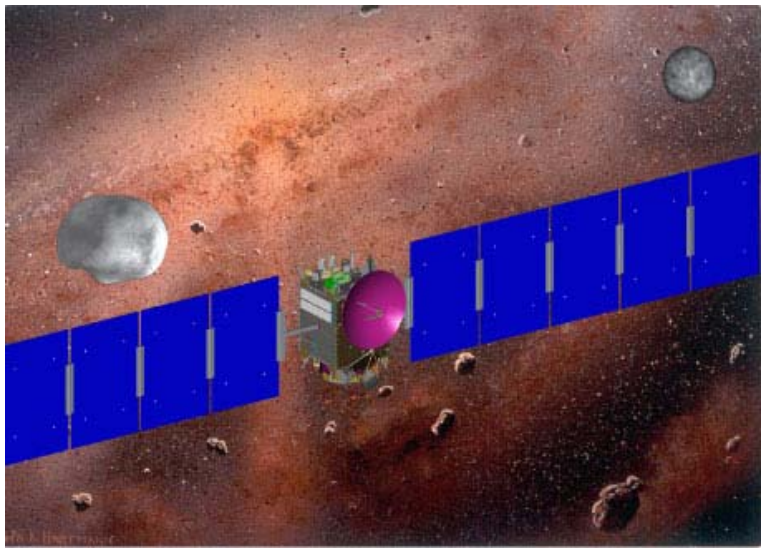
RTEMS – zastosowanie (2)

System naprowadzania rakiet ziemia-powietrze

System umożliwiający realizację szyfrowanej komunikacji bezprzewodowej

RTEMS – zastosowanie (3)

- RTEMS i eksploracja przestrzeni kosmicznej



Satelita Dawn

...The real-time operating system RTEMS is used as basis for a complete instrument control and on-board data processing system, implemented in a sophisticated on-board command language (OCL)...

Herschel - teleskop kosmiczny do dalekich obserwacji

RTEMS

- **System czasu rzeczywistego**

RTEMS został od samego początku zaprojektowany jako system czasu rzeczywistego.

- **System czasu rzeczywistego, czy “Task Manager”**

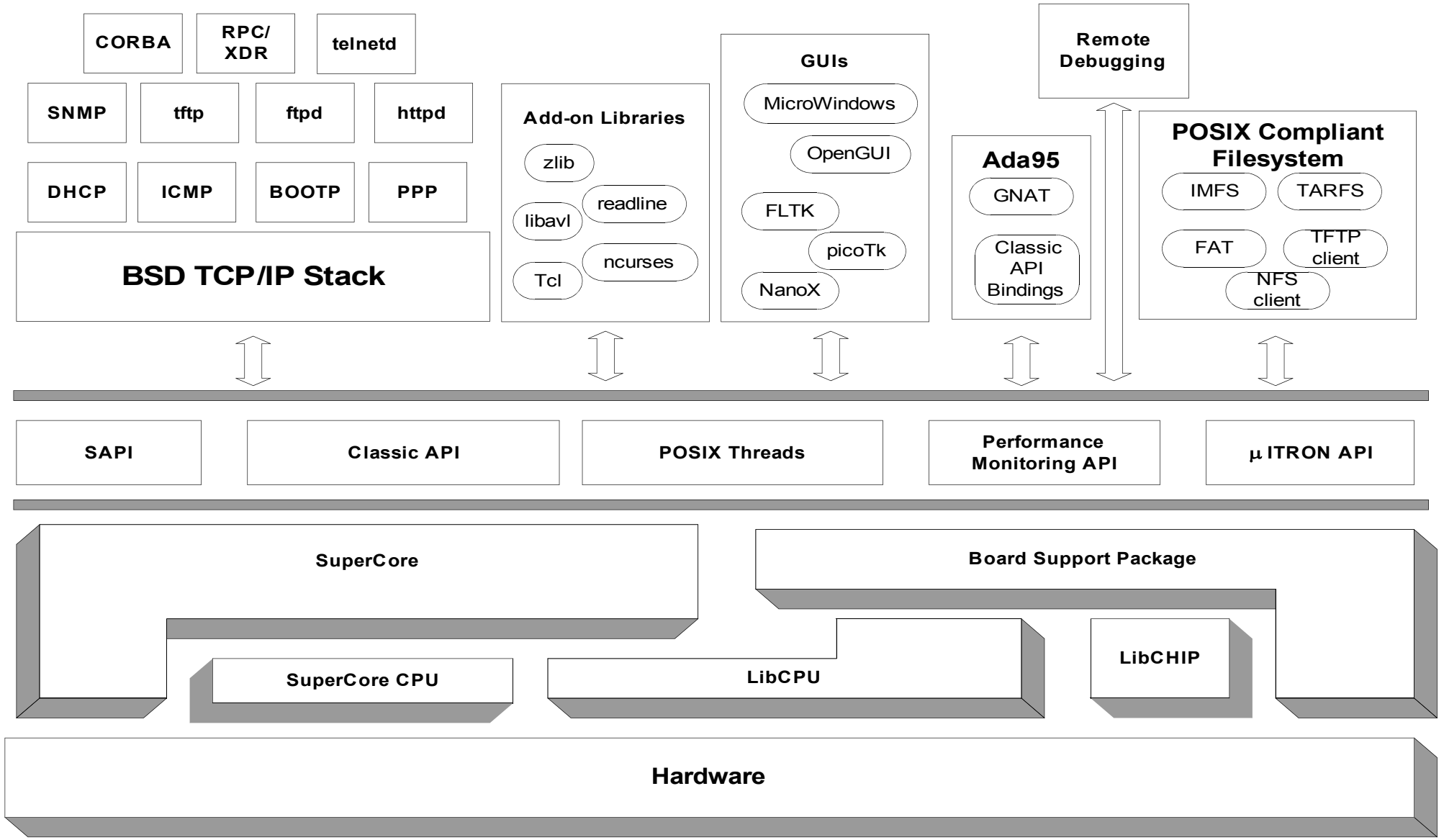
RTEMS może zostać użyty jako prosty manager zadań (~15 kB) lub system operacyjny (>~150 kB).

- **Środowisko uruchomieniowe**
  - Środowisko uruchomieniowe zawiera wszystkie niezbędne narzędzia do kompilacji systemu oraz aplikacji (GNU, Linux, Windows),
  - Kompilator,
  - Debuger,
  - Bogate wsparcie dla różnych procesorów oraz zestawów uruchomieniowych BSP (Board Support Package),
  - Wsparcie techniczne: grupy dyskusyjne, szkolenia, itd...

Charakterystyka systemu RTEMS

- Wyposażony w jądro czasu rzeczywistego,
- Planista odpowiedzialny za przełączanie i synchronizację zadań,
- Komunikacja i synchronizacja międzyprocesowa,
- Skalowalna architektura,
- Obsługa przerwań i wyjątków,
- System wielozadaniowy, wywłaszczanie oparte na zdarzeniach i priorytetach,
- Komunikacja i synchronizacja międzyprocesowa,
- Przenośny – dostępne BSP dla różnych platform sprzętowych,
- System wieloprocesorowy,
- Komunikacja międzyprocesowa,
- Serwer do zdalnego debugowania,
- Shell (telnet, port szeregowy),
- Zarządzanie pamięcią, dynamiczna alokacja pamięci,
- Sterowniki I/O,
- Stos TCP/IP, serwery sieciowe,
- System plików,
- Wsparcie dla języka C, C++, Java,
- Libchip – bibliotek urządzeń peryferyjnych,
- Wsparcie debugowania (JTAG, port szeregowy, Ethernet).

Architektura systemu operacyjnego RTEMS



Natywne API systemu operacyjnego (Classic API)

Oparte na specyfikacji RTEID/ORKID (znacznie prostsze niż POSIX),

RTEMS udostępnia następujące funkcje sterujące zadaniami:

- Semafory (zliczające, synchronizacja warunkowa, muteksy),
- Zdarzenia,
- Kolejki komunikatów,
- Sygnały,
- Bariery,

Przykład użycia API:

```
rtems_status_code
 rtems_semaphore_create( rtems_name name, uint32_t count,
                         rtems_attribute attribute_set,
                         rtems_task_priority priority_ceiling,
                         rtems_id *id )
```

API zgodne z standardem POSIX

- Oparte na specyfikacji POSIX 1003.1b

  RTEMS udostępnia nastepujące funkcje sterujące zadaniami:

  - Semafory,
  - Muteksy,
  - Zmienne warunkowe,
  - Kolejki komunikatów,
  - Sygnały,
  - Bariery,
  - Blokujące operacje odczytu/zapisu.

Przykład użycia API:

```
int sem_init( sem_t *sem, int pshared, unsigned int value )
```

API zgodne z standardem uITRON

- Oparte na specyfikacji uITRON 3.0

  RTEMS udostępnia następujące funkcje sterujące zadaniami:

  - Semafory,
  - Zdarzenia przesyłane przy pomocy sygnałów globalnych (Eventflag),
  - Mailbox,
  - Kolejki komunikatów,

Przykład użycia API:

```
ER pget_blk( VP *p_blk,ID mplid,INT blksz )
```

Środowisko uruchomieniowe

Środowisko uruchomieniowe (ang. Development Environment)

- Oparte na narzędziach GNU (C, C++, Ada, Java, Fortran),
  - GNU debuger,
  - Binutils (ld, nm, etc...),
- Skrypty GNU autoconf dla konfiguracji RTEMS,
- Skrypty Makefile dla aplikacji i sterowników (umożliwiają izolację pomiędzy procesorem, a językiem programowania),
- Newlib
  - Biblioteki języka C dla urządzeń wbudowanych,
- Konfiguracja i instalacja RTEMS odbywa się tak samo jak innych programów korzystających z narzędzi GNU (Linux).

Wspierane języki programowania

- Język C
  - Zawiera bibliotekę standardową,
- C++,
  - Zawiera szablony dostępne w bibliotece standardowej,
- Ada
  - Zawiera pakiety języka Ada,
- Java
  - Dostępny kompilator GNU dla języka Java (gjc) oraz wsparcie dla JVM (Java Virtual Machine),
- Wsparcie dla języków skryptowych
  - Python.

Zarządzanie pamięcią

- Klasyczne API zarządzania pamięcią
  - Partycje pamięci,
  - Regions - alokacja, rezerwacja pamięci o stałej wielkości,
  - Obsługa pamięci dwuportowych,
- Dynamiczna alokacja pamięci zgodna ze standardem ANSI/ISO C
  - Funkcje malloc/free,

Obsługa przerwań

Cechy mechanizmów do obsługi przerwań systemu RTEMS:

- Szybka reakcja na przerwania,
- Obsługa przerwań przy pomocy uchwytów napisanych w języku wyższego poziomu (C/C++),
- Mogą wpływać na zachowanie zadań,
- Przerwania mogą być wyłączane dla minimalizacji czasu wykonania usług czasu rzeczywistego,
- Współpraca ze sterownikami przerwań – przerwania wektorowe, przechwytywanie i rozpoznawanie źródła przerwania.

System operacyjny, dostępne platformy sprzętowe

Głównym celem podczas projektowania systemu RTEMS było zapewnienie łatwej przenośności pomiędzy różnymi platformami sprzętowymi,

- Izolacja warstwy sprzętowej od systemu operacyjnego,
- Otwarty kod źródłowy,
- System dostępny dla wielu różnych procesorów,
- Dostępne pakiety startowe BSP dla różnych systemów komputerowych,
- Wsparcie dla języków C/C++, Ada.

Procesory wspierane przez system RTEMS

- ARM
- Motorola MC680x0
- Motorola MC683xx
- Freescale Coldfire
- Freescale PowerPC
- Intel i386 and above
- Intel i960
- MIPS
- OpenCores OR32
- SPARC
- AMD A29K
- Hewlett-Packard PA-RISC
- Hitachi H8
- Hitachi SH
- Texas Instruments C3x
- Texas Instruments C4x

Cechy systemu operacyjnego RTEMS

- Stos TCP/IP
  - TCP, UDP
  - ICMP, DHCP, RARP
  - RPC, CORBA
  - TFTP, FTPD, HTTPD
- Wsparcie dla systemów plików
  - In-Memory Filesystem (IMFS)
  - TFTP Client Filesystem
  - FTP Client Filesystem
  - FAT Filesystem (IDE and CompactFlash)

Obsługa portów wejścia/wyjścia

- Większość aplikacji wymaga dostępu do portów I/O,
- Obsługa różnych urządzeń dołączonych to tego samego portu,
- Ustandaryzowane API, dostępne funkcje:
  - Initialize,
  - Open,
  - Close,
  - Read,
  - Write,
  - Control.

Obsługa urządzeń peryferyjnych

System RTEMS zapewnia obsługę następujących urządzeń peryferyjnych:

- UART Channels A and B
- Timer General Purpose Timer
- Timer Real Time Clock
- Watchdog Timer
- Control Register
- Memory Control Register
- Interrupt Control

Konfiguracja systemu RTEMS

Konfiguracja systemu RTEMS (1)

Konfiguracja SO RTEMS jest zwykle przechowywana w pliku „system.h".

```
/* configuration information */
#include <bsp.h>                                    /* for device driver prototypes */
/* examplary configuration of RTEMS system */
#define CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER // no clk driver
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER                  // with timer
#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_MAXIMUM_TASKS             4
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_EXTRA_TASK_STACKS         (3 *RTEMS_MINIMUM_STACK_SIZE)
#include <rtems/confdefs.h>                         /* for OS configuration */

#define CONFIGURE_INIT                              /* force OS user-defined configuration */
```

Konfiguracja systemu RTEMS (2)

```
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_LIBIO_MAXIMUM_FILE_DESCRIPTORS 8
#define CONFIGURE_EXECUTIVE_RAM_SIZE (512*1024)
#define CONFIGURE_MAXIMUM_SEMAPHORES            20
#define CONFIGURE_MAXIMUM_TASKS                 20
#define CONFIGURE_MICROSECONDS_PER_TICK         1000        /* 1 millisecond */
#define CONFIGURE_TICKS_PER_TIMESLICE           50          /* 50 milliseconds */

#define CONFIGURE_INIT_TASK_STACK_SIZE (10*1024)
#define CONFIGURE_INIT_TASK_PRIORITY   120
#define CONFIGURE_INIT_TASK_INITIAL_MODES (   RTEMS_PREEMPT | \
                                              RTEMS_NO_TIMESLICE | \
                                              RTEMS_NO_ASR | \
                                              RTEMS_INTERRUPT_LEVEL(0))
```

Konfiguracja systemu RTEMS (3)

/* zawartość pliku konfiguracyjnego system.h */

```
#define CONFIGURE_INIT          // włącza konfigurację systemu RTEMS, plik confdefs.h
                                // konfiguracja przy pomocy dyrektyw preprocesora
```

dyrektywa CONFIGURE_INIT musi występować tylko raz w pliku, który włącza confdefs.h

Jeżeli CONFIGURE_INIT nie jest zdefiniowane system RTEMS budowany jest z minimalna ilością zasobów:

- 1 zadanie,
- minimalna ilość pamięci przeznaczona na
  - stos,
  - stos procedur obsługujących przerwania,
  - stos procedur obsługujących zadania,
  - całego OS,
- brak timerów,
- konsol,
- itd...

Konfiguracja systemu RTEMS (4)

/* przykładowa zawarość pliku confdefs.h */

…

...

```
#ifdef CONFIGURE_INIT
  rtems_driver_address_table Device_drivers[] = {
    #ifdef CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
          CONSOLE_DRIVER_TABLE_ENTRY,
    #endif
    #ifdef CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
          CLOCK_DRIVER_TABLE_ENTRY,
    #endif
}
    #ifdef CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
                #include <rtems/console.h>
    #endif
#endif
```

Konfiguracja systemu RTEMS (5)

```
#include "system.h"
rtems_task Init ( rtems_task_argument argument )
{
        rtems_status_code status;
        rtems_time_of_day time;
        …
        /* create new tasks, timers, queues, semaphores, etc... */
        …
        / * start created tasks, timers, queues, semaphores, etc... */
        …
        /* when everythins goes well exit Init task */
        ...
        status = rtems_task_delete( RTEMS_SELF );
}
```

Zarządzanie zadaniami w systemie RTEMS

Obsługa zadań w systemie RTEMS

- System RTEMS udostępnia mechanizmy pozwalające na zarządzanie zadaniami (wątkami),
- Wszystkie zadania współdzielą wspólną pamięć RAM (wspólna przestrzeń adresowa),
- Każde zadanie posiada własny stos oraz strumienie komunikacyjne (w języku C stdin, stdout, stderr),
- Do zadania przypisany jest priorytet,
- Zadania mogą być:
  - Wywłaszczalne (ang. preemptitive),
  - Wykonywane z podziałem czasu (ang. timeslicing),
  - Obsługują sygnały asynchroniczne,
- Szeregowanie zadań zgodnie z algorytmem Round Robin,
- Obsługa wyjątków związanych z realizacją zadań.

Obsługa zadań w systemie RTEMS

- Zadania w systemie RTEMS opisane są przy pomocy struktury TCB (Task Control Block).
- Podczas inicjalizacji systemu RTEMS rezerwuje miejsce na struktury TCB dla każdego zadania. Liczba zadań ustalana jest przy pomocy makra

  #define CONFIGURE_MAXIMUM_TASKS 3
- Podczas tworzenia zadania wypełniana jest struktura TCB zawierająca:
  - Nazwę zadania,
  - Numer ID identyfikujący dane zdanie,
  - Priorytet zadania,
  - Obecny stan oraz stan zadania podczas uruchamiania,
  - Tryb w jakim zadanie jest wykonywane,
  - Wskaźnik do struktury użytkownika,
  - Wskaźnik do struktury ustalającej zasady szeregowania zadania,
  - Dane opisujące stan zadania zablokowanego,
- Kontekst zadania zapamiętywany jest w TCB podczas przełączania zadania.

Stan zadania

- Zadania w systemie RTEMS mogą znajdować się w jednym z następujących stanów:
  - Executing – zadanie dla którego przydzielono czas CPU,
  - Ready – zadanie oczekujące na przydzielenie czasu CPU,
  - Blocked – zadanie, które dla którego nie można przydzielić czasu CPU,
  - Dormant – zadanie utworzone (wypełniona struktura TCB), jednak nie jest uruchomione (planista nie ma dostępu do struktury TCB),
  - Non-existent – zadanie nie zostało utworzone lub zostało usunięte (brak wpisu w tablicy TCB).
- Przydzielanie czasu procesora realizowane jest w zależności od priorytetu zadania oraz obecnego stanu.

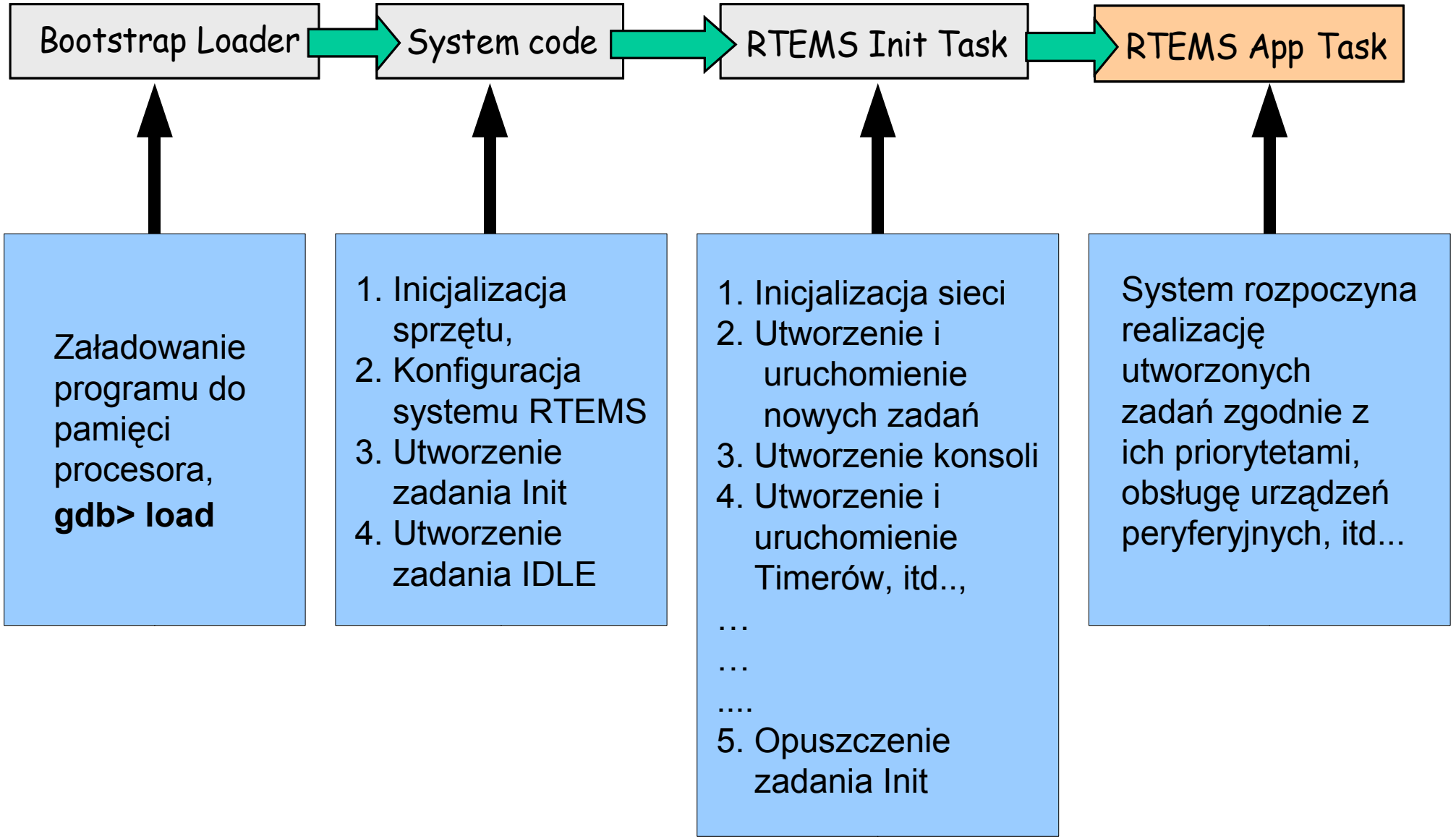
Funkcje obsługujące zadania udostępniane przez RTEMS

- rtems_task_create – utwórz nowe zadanie
- rtems_task_ident – pobierz numer ID zadania
- rtems_task_self – zwróć ID zadania obecnie wykonywanego
- rtems_task_start – uruchom nowe zadanie
- rtems_task_restart – ponownie uruchom zadanie (z pierwotnymi parametrami)
- rtems_task_delete – usuń zadanie
- rtems_task_suspend – uśpij zadanie
- rtems_task_resume – przywróć zadanie
- rtems_task_is_suspended – sprawdź, czy zadanie znajduje się w stanie uśpienia
- rtems_task_set_priority – ustaw priorytet zadania
- rtems_task_mode – zmień tryb pracy danego zadania (preemption, timesllicing, INT)
- rtems_task_get_note – zwróć wskaźnik do tablicy notepad
- rtems_task_set_note – ustal wskaźnik do tablicy notepad
- rtems_task_wake_after – uruchom zadanie po danym czasie (rozdzielczość w tikach)
- rtems_task_wake_when – uruchom zadanie według podanej daty (rozdzielczość ~1 s)
- rtems_task_variable_add – przypisz zmienną do zadania
- rtems_task_variable_get – odczytaj zawartość zmiennej przypisanej do zadania
- rtems_task_variable_delete – usuń zmienną przypisanej do zadania

Zadanie Init

```
rtems_task Init( rtems_task_argument ignored)
{
  rtems_status_code status;

/*        Inicjalizacja wykonana przez użytkownika – utworzenie nowych:
 *                                                              zadań,
 *                                                              timerów,
 *                                                              semaforów, itd...
 */

/*        Usunięcie zadania Init          */

        status = rtems_task_delete(RTEMS_SELF);  // jeżeli nie utworzymy innego zadania system zakończy
                                                 // pracę
  if (status != RTEMS_SUCCESSFUL) {
     fprintf(stderr, "status = %d\n", status);
     printf("Error deleting Init task\n");
     exit(0); }

/*        Zadanie Init zostało prawidłowo usunięte
 *        dalsze wykonanie programu realizowane jest przez utworzone zadania
 *        program nie powinien nigdy dotrzeć do tego miejsca
*/
   exit (0);
 }
```

RTEMS – procedura uruchamiania



Inicjalizacja systemu RTEMS

- Inicjalizacja systemu operacyjnego,
- Inicjalizacja sterowników urządzeń,
- Obsługa biblioteki ANSI/ISO C,
- Przejęcie obsługi zadania Init.

Tworzenie nowego zadania

Do utworzenia nowego zadania wykorzystywana jest funkcja:

```
rtems_status_code rtems_task_create( rtems_name name,
                                     rtems_task_priority  initial_priority,
                                     size_t stack_size,
                                     rtems_mode initial_modes,
                                     rtems_attribute attribute_set,
                                     rtems_id *id );
```

Funkcja może zwrócić jeden z następujących kodów statusu:

- RTEMS_SUCCESSFUL – zadanie utworzone poprawnie
- RTEMS_INVALID_ADDRESS – wskaźnik ID jest nie ustawiony
- RTEMS_INVALID_NAME – błędna nazwa zadania
- RTEMS_INVALID_PRIORITY – błędny priorytet zadania
- RTEMS_MP_NOT_CONFIGURED – system wieloprocesorowy nie jest obsługiwany (ustawiony)
- RTEMS_TOO_MANY – zbyt dużo globalnych obiektów, utworzonych zadań
- RTEMS_UNSATISFIED – zbyt mała pamięć dostępna na stos lub kontekst FP

Nazwa zadania

```
rtems_status_code rtems_task_create( rtems_name name,
                                     rtems_task_priority  initial_priority,
                                     size_t stack_size,
                                     rtems_mode initial_modes,
                                     rtems_attribute attribute_set,
                                     rtems_id *id );
```

- Nazwa zadania powinna jednoznacznie identyfikować zadanie,
- Nazwa zadania może być zrealizowana przy pomocy liczby 32 bitowej bez znaku lub może składać się z 4 znaków ASCII,
- Do tworzenia nazwy zadania ASCII należy użyć funkcji rtems_build_name()

```
rtems_object_name my_name;
my_name = rtems_build_name( 'L', 'I', 'T', 'E' );
```

- Identyfikacja nazwy zazania:

```
void print_name(rtems_id the_object) {
        char buffer[10]; /* name assumed to be 10 characters or less */
        char *result;
        result = rtems_object_get_name( id, sizeof(buffer), buffer );
        printk( "ID=0x%08x name=%s\n", id, ((result) ? result : "no name") );
}
```

Makra do tworzenia nazwy zadania

System RTEMS udostępnia proste makro umożliwiające utworzenie nazwy zadania na podstawie 4 znaków ASCII:

```
rtems_object_name my_name;
my_name = rtems_build_name( 'L', 'I', 'T', 'E' );
```

```
#define  rtems_build_name( _C1, _C2, _C3, _C4 ) \
  ( (uint32_t)(_C1) << 24 | \
    (uint32_t)(_C2) << 16 | \
    (uint32_t)(_C3) << 8  | \
    (uint32_t)(_C4) )
```

Priorytety zadań

```
rtems_status_code rtems_task_create( rtems_name name,
                                     rtems_task_priority  initial_priority,
                                     size_t stack_size,
                                     rtems_mode initial_modes,
                                     rtems_attribute attribute_set,
                                     rtems_id *id );
```

- System RTEMS obsługuje 255 poziomów priorytetów, 1 – priorytet najwyższy, 255 najniższy, zadanie Init ma domyślnie przydzielony priorytet =1,
- Ten sam priorytet może zostać przypisany do kilku zadań,
- Priorytet nadawany jest podczas tworzenia zadania, można go w każdej chwili zmienić

```
rtems_status_code rtems_task_set_priority( rtems_id id, rtems_task_priority  new_priority,
  rtems_task_priority *old_priority);
```

Przykład:

```
status = rtems_task_create( task_name, 50, ...);
#define CONFIGURE_INIT_TASK_PRIORITY      1    /* zmiana domyśl. priorytetu zad. Init */
```

Rozmiar przydzielonego stosu

```
rtems_status_code rtems_task_create( rtems_name name,
                                     rtems_task_priority  initial_priority,
                                     size_t stack_size,
                                     rtems_mode initial_modes,
                                     rtems_attribute attribute_set,
                                     rtems_id *id );
```

System RTEMS posiada zdefiniowane stałe do obsługi rozmiaru stosu:

- RTEMS_MINIMUM_STACK_SIZE – minimalny rozmiar stosu
- RTEMS_CONFIGURED_MINIMUM_STACK_SIZE – rozmiar zdefiniowany przez programistę

Parametry określające tworzone zadanie

```
rtems_status_code rtems_task_create( rtems_name name,
                                     rtems_task_priority  initial_priority,
                                     size_t stack_size,
                                     rtems_mode initial_modes,
                                     rtems_attribute attribute_set,
                                     rtems_id *id );
```

- RTEMS_PREEMPT – aktywacja trybu wywłaszczania (domyślnie)
- RTEMS_NO_PREEMPT – wyłącza tryb wywłaszczania
- RTEMS_NO_TIMESLICE – wyłącza tryb z podziałem czasu (domyślnie)
- RTEMS_TIMESLICE – włącza tryb z podziałem czasu, zadania z równymi priorytetami,
- RTEMS_ASR – włącza obsługę sygnałów ASR, wymaga proc. obsługi ASR (domyślnie)
- RTEMS_NO_ASR - włącza obsługę sygnałów ASR
- RTEMS_INTERRUPT_LEVEL(0) – włącza obsługę wszystkich przerwań (domyślnie)
- RTEMS_INTERRUPT_LEVEL(n) - włącza obsługę wszystkich z poziomem większym od n
- RTEMS_DEFAULT_MODES – włącza parametry domyślne

Atrybuty zadań

```
rtems_status_code rtems_task_create( rtems_name name,
                                     rtems_task_priority  initial_priority,
                                     size_t stack_size,
                                     rtems_mode initial_modes,
                                     rtems_attribute attribute_set,
                                     rtems_id *id );
```

Dodatkowe atrybuty wykorzystywane podczas tworzenia zadań:

- RTEMS_FLOATING_POINT – przydziela dodatkową pamięć dla TLB używaną podczas przełączania kontekstu FPU
- RTEMS_NO_FLOATING_POINT – brak pamięci na TLB dla FPU
- RTEMS_LOCAL – zadanie lokalne (default)
- RTEMS_GLOBAL – zadanie globalne

Przykład użycia atrybutów:

```
Status = rtems_task_create( task_name, 50, ..., RTEMS_GLOBAL |
                                                RTEMS_FLOATING_POINT, task_id);
```

Wskaźnika to struktury opisującej zadanie

```
rtems_status_code rtems_task_create( rtems_name name,
                                     rtems_task_priority  initial_priority,
                                     size_t stack_size,
                                     rtems_mode initial_modes,
                                     rtems_attribute attribute_set,
                                     rtems_id *id );
```

```
rtems_id   task_id;
rtems_id   tasks_id[ 10 ];
```

```
status = rtems_task_create (main_task, ..., &task_id );

status = rtems_task_create (task_1, ..., &tasks_id [1]);
```

Wskaźnik do funkcji obsługującej zadanie

```
status = rtems_task_start( tasks_id[0], task_body, 1 );

rtems_task task_body(rtems_task_argument argument)
{
        uint32_t        local_variable      /* unsigned int type */
  while (1){                                /* infinite loop */

                        /* application code goes here */

        local_variable = argument;          /* use argument, unsigned int type, can be pointer to
                                               array of parameters, etc... */
  }
}
```

Parametry określające tworzone zadanie

```
rtems_status_code rtems_task_start( rtems_id id, rtems_task_entry    entry_point,
                                    rtems_task_argument argument );
```

- RTEMS_SUCCESSFUL – zadanie utworzone poprawnie
- RTEMS_INVALID_ADDRESS – błędny wskaźnik do funkcji obsługującej zadania
- RTEMS_INVALID_ID – nie ustawiony wskaźnik ID
- RTEMS_INCORRECT_STATE – zadanie w stanie dormant
- RTEMS_ILLEGAL_ON_REMOTE_OBJECT – nie można uruchomić zadania zdalnego w systemie wieloprocesorowym

Przykład funkcji tworzącej zadanie

```
rtems_task Init( rtems_task_argument ignored)
{
  rtems_status_code          status;
  rtems_id                   tid, tasks_id[MAX_TASKS];

  status = rtems_task_create(
                    rtems_build_name('T','S','K','1'), 10, RTEMS_MINIMUM_STACK_SIZE,
                    RTEMS_DEFAULT_MODES, RTEMS_DEFAULT_ATTRIBUTES, &tasks_id[0] );

        if (status != RTEMS_SUCCESSFUL) { ....}
  status = rtems_task_start(tasks_id[0], task_handler, (rtems_task_argument)tsk1_msg);
   `    if (status != RTEMS_SUCCESSFUL) {
              fprintf(stderr, "status = %d\n", status);
              printf("Error starting task 1\n");
              exit(0);
  }
        rtems_task_delete( RTEMS_SELF );
        exit (1);
}
```

Identyfikacja zadań

```
rtems_id              tid;
rtems_status_code     status;
rtems_task_priority   old_priority;
rtems_mode            old_mode;
uint32_t              task_index;
rtems_id              tid, tasks_id[MAX_TASKS];

/* return ID of the task */
status = rtems_task_ident( RTEMS_SELF, RTEMS_SEARCH_ALL_NODES, &tid );
status = rtems_task_ident( rtems_build_name( 'L', 'I', 'T', 'E' ),
                           RTEMS_SEARCH_ALL_NODES, &tid );
/* return index of task in TCB */
task_index = task_number( tid );
```

Parametr ID jest potrzebny do przeprowadzenia większości operacji związanych z obsługą zadań, np:

```
status = rtems_task_restart( tid, NULL);
status = rtems_task_suspend( tid );
```

Modyfikacja parametrów zadań

```
rtems_id            tid;
rtems_status_code   status;
rtems_task_priority old_priority;
rtems_mode          old_mode;
uint32_t            task_index;
rtems_id            tid, tasks_id[MAX_TASKS];


/* change task priority */
status = rtems_task_set_priority( RTEMS_SELF, RTEMS_MAXIMUM_PRIORITY - 1,
                                  &old_priority );
status = rtems_task_set_priority( tid, RTEMS_MAXIMUM_PRIORITY - 1,
                                  &old_priority );
/* change the current task mode */
status = rtems_task_mode( RTEMS_PREEMPT,  RTEMS_PREEMPT_MASK, &old_mode );
```

Zestaw przykładów...

Przykładowe programy można znaleźć w katalogu zawierającym źródła RTEMS-a:

...\rtems-4.10\testsuites\samples

\sptests

\support

\tmtest

\...

Warto zajrzeć do:

\rtems-4.10\testsuites\samples\hello

\rtems-4.10\testsuites\samples\minimum

\rtems-4.10\testsuites\samples\ticker

\rtems-4.10\testsuites\samples\iostream

Zarządzanie czasem w systemie RTEMS

Timer manager

Clock manager

Timer Manager

System RTEMS dostarcza zestaw funkcji (Timer Manager) pełniących rolę timera programowego. Aplikacje pracujące pod kontrolą systemu operacyjnego nie powinny bezpośrednio korzystać z timera sprzętowego (za wyjątkiem specjalizowanych operacji, np. PWM). Timer jest obiektem, który umożliwia obsługę zadań uzależnionych od czasu. Timer może zostać wykorzystany do odmierzania czasu.

Funkcje wykorzystywane do obsługi timera:

rtems_timer_create – funkcja tworząca obiekt timera

rtems_timer_ident – funkcja zwraca wskaźnik do obiektu timera (timer ID)

rtems_timer_cancel – funkcja odwołuje ustawiony timer

rtems_timer_delete - funkcja niszczy obiekt timera, zwalnia zasoby

rtems_timer_fire_after - funkcja generująca zadane opóźnienie

rtems_timer_fire_when - funkcja generująca opóźnienie zależne od daty

rtems_timer_initiate_server - funkcja tworząca obiekt serwera timera (tworzy zadanie serwera)

rtems_timer_server_fire_after - funkcja generująca zadane opóźnienie (serwer)

rtems_timer_server_fire_when - funkcja generująca opóźnienie zależne od warunku (serwer)

rtems_timer_reset - funkcja zerująca dany timer

Utworzenie przykładowego timera

rtems_status_code **rtems_timer_create**(rtems_name name, rtems_id *id);

- Tworzy obiekt timera (wypełnia strukturę timera, zwraca wskaźnik do obiektu timera).
- Należy wcześniej zarezerwować miejsce przeznaczone na struktury timera.

Kody zwracane przez funkcje rtems_timer_create:

RTEMS_SUCCESSFUL - timer utworzony poprawnie

RTEMS_INVALID_ADDRESS – wskaźnik id nie jest ustawiony (NULL)

RTEMS_INVALID_NAME – błędna nazwa timera

RTEMS_TOO_MANY – zbyt dużo utworzonych obiektów timera

```
#define CONFIGURE_MAXIMUM_TIMERS      10
```

Funkcje POSIX związane z timerem

```
usleep (1.000.000 );          /* usec parameter must be smaller or equal to than 1.000.000 */
nanosleep(100 );              /* nanoseconds field must be in the range 0 to 999.999.999 */
sleep( 1 );                   /* parameter in seconds */
```

Identyfikacja utworzonego timera

rtems_status_code **rtems_timer_ident**(rtems_name name, rtems_id *id);

Funkcja zwraca wskaźnik do timera wskazanego przy pomocy unikalnej nazwy.

Kody zwracane przez funkcję rtems_timer_create:

RTEMS_SUCCESSFUL – timer zidentyfikowany prawidłowo

RTEMS_INVALID_ADDRESS - wskaźnik id nie jest ustawiony (NULL)

RTEMS_INVALID_NAME – nie znaleziono timera o takiej nazwie

Odwołanie wyzwolonego timera

rtems_status_code **rtems_timer_cancel** (rtems_name name, rtems_id *id);

- Funkcja odwołuje ustawiony timer.
- Nie nastąpi wywołanie funkcji skojarzonej z danym timerem po upływie zadanego czasu.
- Timer można wyzwolić ponownie przy pomocy funkcji timer_fire_after lub timer_fire_when.

Kody zwracane przez funkcję rtems_timer_create:

RTEMS_SUCCESSFUL – timer odwołany prawidłowo

RTEMS_INVALID_ADDRESS - wskaźnik id nie jest ustawiony (NULL)

Usunięcie obiektu timera

rtems_status_code **rtems_timer_delete** (rtems_name name, rtems_id *id);

- Funkcja odwołuje i zwalnia zasoby danego timera.
- Zwolnione zasoby przekazywane są do systemu RTMS.
- Można utworzyć nowy timer wykorzystując zwolnione zasoby.

Kody zwracane przez funkcję rtems_timer_create:

RTEMS_SUCCESSFUL – timer usunięty prawidłowo

RTEMS_INVALID_ADDRESS - wskaźnik id nie jest ustawiony (NULL)

Uruchomienie timera (1)

rtems_status_code **rtems_timer_fire_after**(rtems_id id, rtems_interval ticks,
rtems_timer_service_routine_entry routine, void *user_data)

- Funkcja powoduje inicjalizację i uruchomienie timera.
- Jeżeli timer już działa, zostaje odwołany i ponownie zainicjalizowany.
- Funkcja obsługująca timer jest wywoływana po upłynięciu zadanej liczby taktów.
- Po uaktywnieniu timera następuje skok do funkcji obsługującej dany timer (routine).

Kody zwracane przez funkcje rtems_timer_fire_after:

RTEMS_SUCCESSFUL – timer zainicjalizowany prawidłowo

RTEMS_INVALID_ADDRESS – nie ustawiony wskaźnik do funkcji obsługującej timer (routine)

RTEMS_INVALID_ID – nie ustawiony wskaźnik ID

RTEMS_INVALID_NUMBER – błędny numer ticks określający liczbę taktów

Uruchomienie timera (2)

rtems_status_code **rtems_timer_fire_when**(rtems_id id, rtems_time_of_day *wall_time, rtems_timer_service_routine_entry routine, void *user_data)

- Funkcja powoduje inicjalizację i uruchomienie timera.
- Jeżeli timer już działa, zostaje odwołany i ponownie zainicjalizowany.
- Funkcja obsługująca timer jest wywoływana, gdy zostanie osiągnięty czas określony przez strukturę wall_time.
- Po uaktywnieniu timera następuje skok do funkcji obsługującej dany timer (routine).

Kody zwracane przez funkcje rtems_timer_fire_when:

RTEMS_SUCCESSFUL – timer zainicjalizowany prawidłowo

RTEMS_INVALID_ADDRESS – nie ustawiony wskaźnik do funkcji obsługującej timer (routine)

RTEMS_INVALID_ADDRESS – wskaźnik struktury TOD wall_time nie ustawiony

RTEMS_INVALID_ID – nie ustawiony wskaźnik ID

RTEMS_INVALID_CLOCK – błędny format czasu podany przez strukturę TOD

Inicjalizacja serwera obsługującego zadania

- RTEMS dostarcza serwer (zadanie) umożliwiający czasowe wstrzymywanie zadań lub uruchamianie ich w określonym czasie.
- Serwer pracuje jako zadanie niewywłaszczalne o najwyższym priorytecie (można traktować je jako przerwanie niskiego poziomu).
- Serwer umożliwia obsługę timerów obsługiwanych podobnie jak zadania (rtems_timer_server_fire_after lub rtems_timer_server_fire_when).
- Zaletą obsługi timera bazującego na zadaniach jest wykonywania operacji zmiennoprzecinkowych wymagających dodatkowej przestrzeni na przełączenie kontekstu, które nie są obsługiwane przez przerwania.

rtems_status_code **rtems_timer_initiate_server**(uint32_t priority, uint32_t stack_size,
tems_attribute attribute_set);

Kody zwracane przez funkcje rtems_timer_create:

RTEMS_SUCCESSFUL – serwer timera utworzony poprawnie

RTEMS_TOO_MANY – zbyt dużo utworzonych obiektów timera

Timer - przykład

```
rtems_status_code status;
rtems_id timer_ID;
status = rtems_timer_initiate_server(RTEMS_TIMER_SERVER_DEFAULT_PRIORITY,
                                     RTEMS_MINIMUM_STACK_SIZE, RTEMS_DEFAULT_ATTRIBUTES);
if (status != RTEMS_SUCCESSFUL){
        rtems_panic("Can't create timer server: %s", rtems_status_text(status));
}
status = rtems_timer_create(rtems_build_name('T','I','M','0'), &timer_ID);
if (status != RTEMS_SUCCESSFUL){
        rtems_panic("Can't create timer: %s", rtems_status_text(status));
}
status = rtems_timer_server_fire_after(timer_ID, 1000/rtems_configuration_get_milliseconds_per_tick(), timer_routine, 0);
                                    // system tick 64 ms
if (status != RTEMS_SUCCESSFUL){
        rtems_panic("Can't start timer: %s", rtems_status_text(status));
}
rtems_timer_service_routine timer_routine(rtems_id id, void *ud)
{
        GlobalTimerCounter++;
status = rtems_timer_server_fire_after(timer_id, 1000/rtems_configuration_get_milliseconds_per_tick(),.....
}
```

Przydatne funkcje

```
MsPerTick=rtems_configuration_get_milliseconds_per_tick();
  printf( "\n\n*** ms per system  tick: %lu\n", MsPerTick );



rtems_task_wake_after (15); /* Task sleepfor 15 ticks => 1 tick = 64 ms => 1s */;
lub
sleep (1);



#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_TIMER_DRIVER
```

Clock Manager

Clock Manager

System RTEMS dostarcza zestaw funkcji (Clock Manager) realizujących funkcję kalendarza:

- Funkcje operują na zdarzeniach związanych czasem i datą
- Funkcje wykorzystywane do obsługi timera:

rtems_clock_set – ustaw datę i czas
rtems_clock_get – pobierz datę i aktualny czas (zgodnie z przekazanym formatem)
rtems_clock_get_tod - pobierz datę i aktualny czas (format TOD, Time Of Day)
rtems_clock_get_tod_timeval - pobierz data i aktualny czas (interval format)
rtems_clock_get_seconds_since_epoch - pobierz liczbę sekund, format POSIX
rtems_clock_get_ticks_per_second – pobierz liczbę taktów na sekundę
rtems_clock_get_ticks_since_boot – pobierz liczbę taktów od uruchomienia systemu
rtems_clock_get_uptime - pobierz czas od uruchomienia systemu
rtems_clock_set_nanoseconds_extension – zainstaluj moduł nanoseconds
rtems_clock_tick – wygeneruj kolejny „takt” czasu (dekrementacja liczników)

- Instalacja sterownika do timera lub RTC

```
#define CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
```

Clock Manager (2)

- System zegara wykorzystuje okresowe przerwania generowane przez timer (inkrementacja zmiennej rtems_clock_tick).
- Zamiast timera można wykorzystać zegar czasu rzeczywistego RTC (Reak Time Clock)
- Czas, który mierzony jest w taktach (ang. tick),
- Długość pojedynczego taktu podawana jest w mikrosekundach, które określana jest w pliku konfiguracyjnym
- Czas przeznaczony na realizację pojedynczego zadania (time_slice) liczony jest również w taktach rtems_clock_tick (np. 1000 taktów).

Clock Manager (3)

- Dane opisujące kalendarz przechowywane są w strukturze TOD:

```
struct rtems_tod_control {
        uint32_t year;          /* greater than 1987 */
        uint32_t month;         /* 1 - 12 */
        uint32_t day;           /* 1 - 31 */
        uint32_t hour;          /* 0 - 23 */
        uint32_t minute;        /* 0 - 59 */
        uint32_t second;        /* 0 - 59 */
        uint32_t ticks;         /* elapsed between seconds */
 };
typedef struct rtems_tod_control rtems_time_of_day;
```

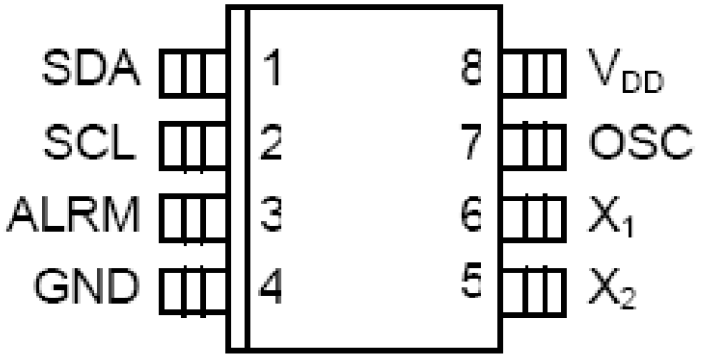
- Istnieje również możliwość odczytania czasu zgodnego ze standardem POSIX (liczonego od 1 stycznia 1970)

```
typedef struct {
  uint32_t seconds;       /* seconds since RTEMS epoch*/
  uint32_t microseconds;  /* since last second        */
} rtems_clock_time_value;
```

Zegar czasu rzeczywistego

Cechy układu DS1629:

- Zegar czasu rzeczywistego,
- Pomiar temperatury -55 – 125 C,
- Rozdzielczość termometru: 9 bitów,
- Dokładność termometru +/- 2 C,
- Układ termostatu,
- 32 bajty pamięci SRAM,
- Zasilanie 2,2 – 5,5 V,
- Interfejs zgodny ze standardem I2C (400 kHz).



DS1629S 8-Pin SOIC
(150-mil)

Rejestry układu RTC

- Układ DS1629 posiada szereg rejestrów przechowujących odmierzany czas, które mapowane są na pamięć

BYTE ADDRESS	BIT 7 MSb	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSb	BYTE RANGE
00h	CH	10 SECONDS			SECONDS				00–59
01h	0	10 MINUTES			MINUTES				00–59
02h	0	12 MODE / 24 MODE	AM/PM / 10 HOURS	10 HOURS	HOURS				01–12 / 00–23
03h	0	0	0	0	0	DAY			01–07
04h	0	0	10 DATE		DATE				01–31*
05h	0	0	0	10 MONTH	MONTH				01–12
06h	10 YEAR				YEAR				00–99

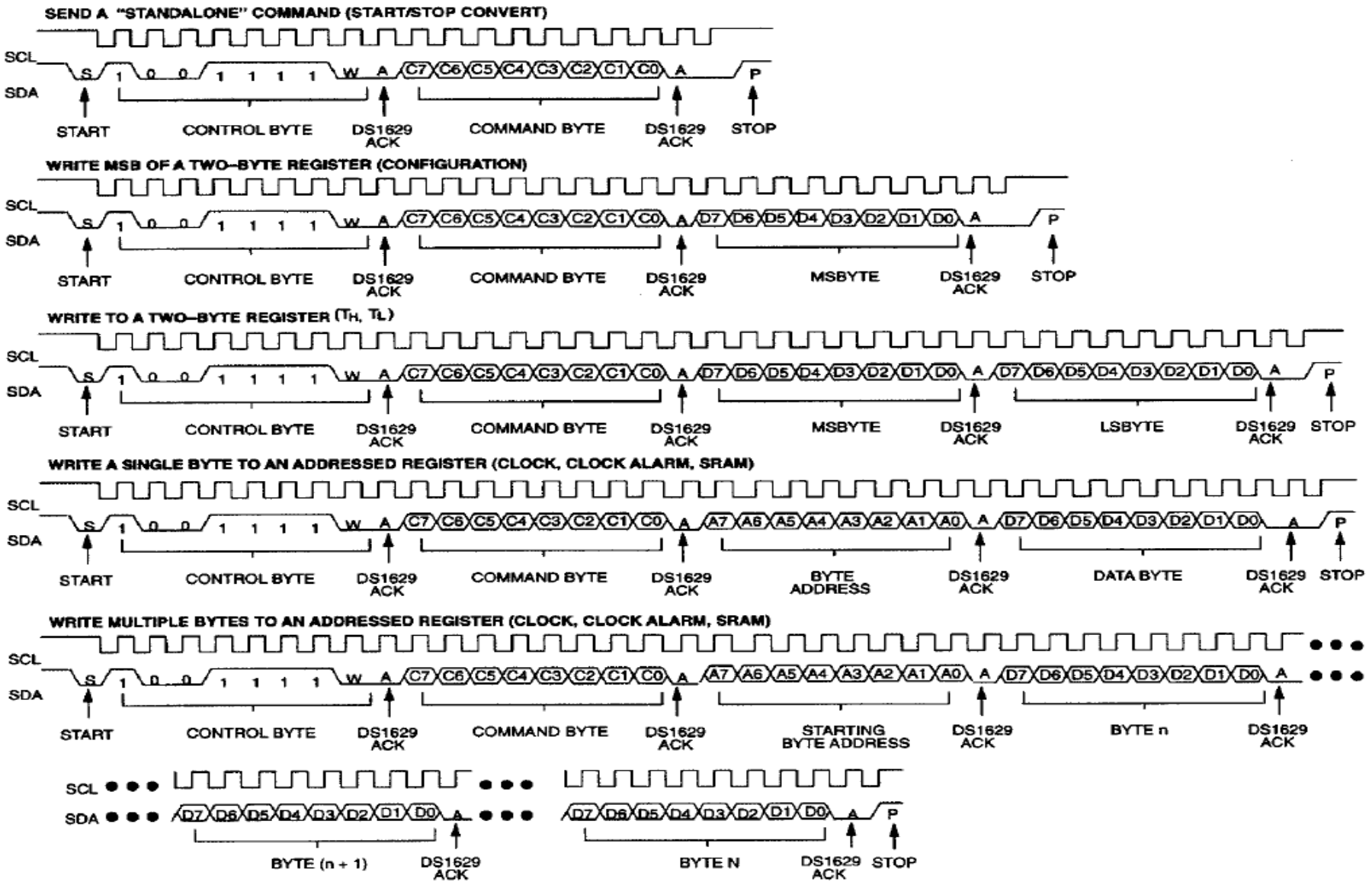
* DATE BYTE MAXIMUM VALUE RANGES FROM 28 TO 31, DEPENDING ON MONTH AND YEAR

Rozkazy sterujące zegarem RTC

DS1629 Command Set Table 8

INSTRUCTION	PROTOCOL	DESCRIPTION	2-wire bus data after issuing protocol	NOTES
CONFIGURATION / MEMORY COMMANDS				
Access Configuration	ACh	Writes to 8-bit configuration register	1 data byte	1, 5
		Reads from configuration/status register	1 or 2 data bytes	
Access Memory	17h	Writes to SRAM array	Starting Address+N-bytes	1, 2
		Read from SRAM array	Starting Address+N-bytes	
THERMOMETER COMMANDS				
Start Convert T	EEh	Initiates temperature conversion(s)	Idle	3
Stop Convert T	22h	Terminates continuous conversions	Idle	3
Read Temperature	AAh	Reads Temperature Register	Read 1 or 2 data bytes	4
Read Counter	A8h	Reads COUNT_REMAIN	Read 1 data byte	
Read Slope	A9h	Reads COUNT_PER_C	Read 1 data byte	
Access TH	A1h	Writes to/Reads from TH register	Write 2 data bytes Read 1 or 2 data bytes	1, 5
Access TL	A2h	Writes to/Reads from TL register	Write 2 data bytes Read 1 or 2 data bytes	1, 5
CLOCK COMMANDS				
Access Clock	C0h	Sets/Reads Clock	Starting Address + N-bytes	1, 2
Access Clock Alarm	C7h	Sets/ Reads Clock Alarm	Starting Address + N-bytes	1, 2

Transmisja z wykorzystaniem interfejsu I2C



Rejestru sterujące funkcją alarm

- Układ można wykorzystać do wygenerowania przerwania w momencie, gdy czas odmierzany przez RTC zrówna się z czasem alarmu.

BYTE ADDRESS	BIT 7 MSb	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSb	BYTE RANGE
00h	0	10 SECONDS			SECONDS				00–59
01h	0	10 MINUTES			MINUTES				00–59
02h	0	0	AM/PM / 10 HOURS	10 HOURS	HOURS				01–12 / 00–23
03h	0	0	0	0	0	DAY			01–07

CLOCK COMMANDS				
Access Clock	C0h	Sets/Reads Clock	Starting Address + N-bytes	1, 2
Access Clock Alarm	C7h	Sets/ Reads Clock Alarm	Starting Address + N-bytes	1, 2

Clock Manager (4)

- Z każdym zadaniem skojarzony jest jeden zegar wykorzystywany do uśpienia lub opóźnienia zadania:
  - rtems_task_wake_after
  - rtems_task_wake_when

rtems_status_code rtems_task_wake_after(rtems_interval ticks);

Blokuje wykonanie zadania na określony czas podanych w taktach;

rtems_status_code rtems_task_wake_when(rtems_time_of_day *time_buffer);

Blokuje wykonanie zadania do momentu osiągnięcia daty podanej jako parametr (TOD);

RTEMS_SUCCESSFUL – zadanie przywrócone poprawnie (znajduje się w kolejcie zadań gotowych do wykonania)

RTEMS_INVALID_ADDRESS – wskaźnik time_buffer nie jest ustawiony

RTEMS_INVALID_TIME_OF_DAY – błąd w strukturze TOD

RTEMS_NOT_DEFINED – data lub czas nie jest ustawiona

Funkcje do obsługi kalendarza (1)

rtems_status_code rtems_clock_set(rtems_time_of_day *time_buffer);

Funkcja sprawdza poprawność przekazanych danych oraz ustawia kalendarz.

Kody zwracane przez funkcję:

- RTEMS_SUCCESSFUL – data ustawiona prawidłowo
- RTEMS_INVALID_ADDRESS – wskaźnik time_buffer nie jest ustawiony
- RTEMS_INVALID_CLOCK – błędne wartość w polu struktury rtems_time_of_day (data przed rokiem 1988 generuje błąd)

Inicjalizacja systemu RTEMS wymaga ustawienia daty (np. przepisanie daty z zegara RTC podłączonego przez interfejs I2C/SPI)

Funkcje do obsługi kalendarza (2)

rtems_status_code rtems_clock_get(rtems_clock_get_options option, void *time_buffer);

Funkcja zwraca aktualny czas zgodnie z podanym polem „options”, możliwe warianty:

- RTEMS_CLOCK_GET_TOD – kalendarz: format TOD (rtems_time_of_day *)
- RTEMS_CLOCK_GET_SECONDS_SINCE_EPOCH – czas od epoki POSIX (rtems_interval *)
- RTEMS_CLOCK_GET_TICKS_SINCE_BOOT – czas od momentu uruchomienia systemu RTEMS (rtems_interval *)
- RTEMS_CLOCK_GET_TICKS_PER_SECOND – liczba taktów na sekundę (rtems_interval *)
- RTEMS_CLOCK_GET_TIME_VALUE – czas: format POSIX, (rtems_clock_time_value *)

Kody zwracane przez funkcję:

- RTEMS_SUCCESSFUL – zwrócono prawidłowy czas i datę
- RTEMS_NOT_DEFINED – czas i data nie są ustawione
- RTEMS_INVALID_ADDRESS – wskaźnik time_buffer nie jest ustawiony

Funkcje do obsługi kalendarza (3)

rtems_clock_get_tod - pobierz datę i aktualny czas (format TOD, Time Of Day)

rtems_clock_get_tod_timeval - pobierz datę i aktualny czas (przeliczoną na liczbę sekund)

rtems_clock_get_seconds_since_epoch - pobierz liczbę sekund, format POSIX

Kody zwracane przez funkcję:

- RTEMS_SUCCESSFUL – zwrócono prawidłowy czas i datę
- RTEMS_NOT_DEFINED – czas i data nie są ustawione
- RTEMS_INVALID_ADDRESS – wskaźnik time_buffer nie jest ustawiony

Konfiguracja zegara

```
#define CONFIGURE_MICROSECONDS_PER_TICK        1000      /* 1 millisecond */
#define CONFIGURE_TICKS_PER_TIMESLICE          50        /* 50 milliseconds */
```

rtems_clock_get_ticks_per_second () – pobierz liczbę taktów na sekundę

Funkcje do obsługi kalendarza (4)

rtems_status_code
rtems_clock_set_nanoseconds_extension(rtems_nanoseconds_extension_routine routine);

- Funkcja instaluje mechanizm dokładnego pomiaru czasu (rzędu nanosekund).
- Funkcja modyfikuje zachowanie sterownika od timera.
- Wymaga użycia wskaźnika do funkcji zwracającej liczbę nanosekund od ostatniego taktu.

Kody zwracane przez funkcję:

- RTEMS_SUCCESSFUL – funkcja zakończona sukcesem
- RTEMS_INVALID_ADDRESS – wskaźnik routine nie jest ustawiony

uint32_t **bsp_clock_nanoseconds_since_last_tick**(**void**);

Komunikacja i synchronizacja w systemie RTEMS

Semafory a system RTEMS

RTEMS udostępnia mechanizmy potrzebne do synchronizacji zadań oraz ochrony sekcji krytycznych. Cechy dostępnych semaforów:

- Semafory zliczające Dijkstra,
- Semafory zgodne ze standardem Posix 1003.1b oraz 1003.1d (timeout),
- RTEMS udostępnia semafory binarne (przyjmuje wartości 0, 1) oraz zliczające (wartości dodatnie).
- Semafory binarne wykorzystywane są do ochrony pojedynczego zasobu.
- Semafory zliczające wykorzystywane są do ochrony kilku różnych zasobów.

Problemy związane z semaforami:

- Inwersja priorytetów,
- Dziedziczenie priorytetów (lokalne, binarne semafory),
  - Pozwalają wyeliminować efekt inwersji priorytetów, spowalniają działanie SO,
- Semafory z pułapem priorytetów (ang. priority ceiling,lokalne, binarne semafory),
  - Szybsze działanie niż sem. dziedziczące lecz bardziej obciążają programistę.

Semafore manager

RTEMS udostępnia mechanizmy potrzebne do synchronizacji zadań oraz ochrony sekcji krytycznych:

W systemie RTEMS dostępne są następujące funkcje:

- SEMAPHORE_CREATE – utwórz semafor
- SEMAPHORE_IDENT – pobierz ID semafora na podstawie jego nazwy
- SEMAPHORE_DELETE – usuń semafor
- SEMAPHORE_OBTAIN – „opuść”, ustaw semafor
- SEMAPHORE_RELEASE – „podnieś”, zwolnij semafor
- SEMAPHORE_FLUSH – odblokuj wszystkie zadania oczekujące na semafor

Atrybuty sterujące semaforami (1)

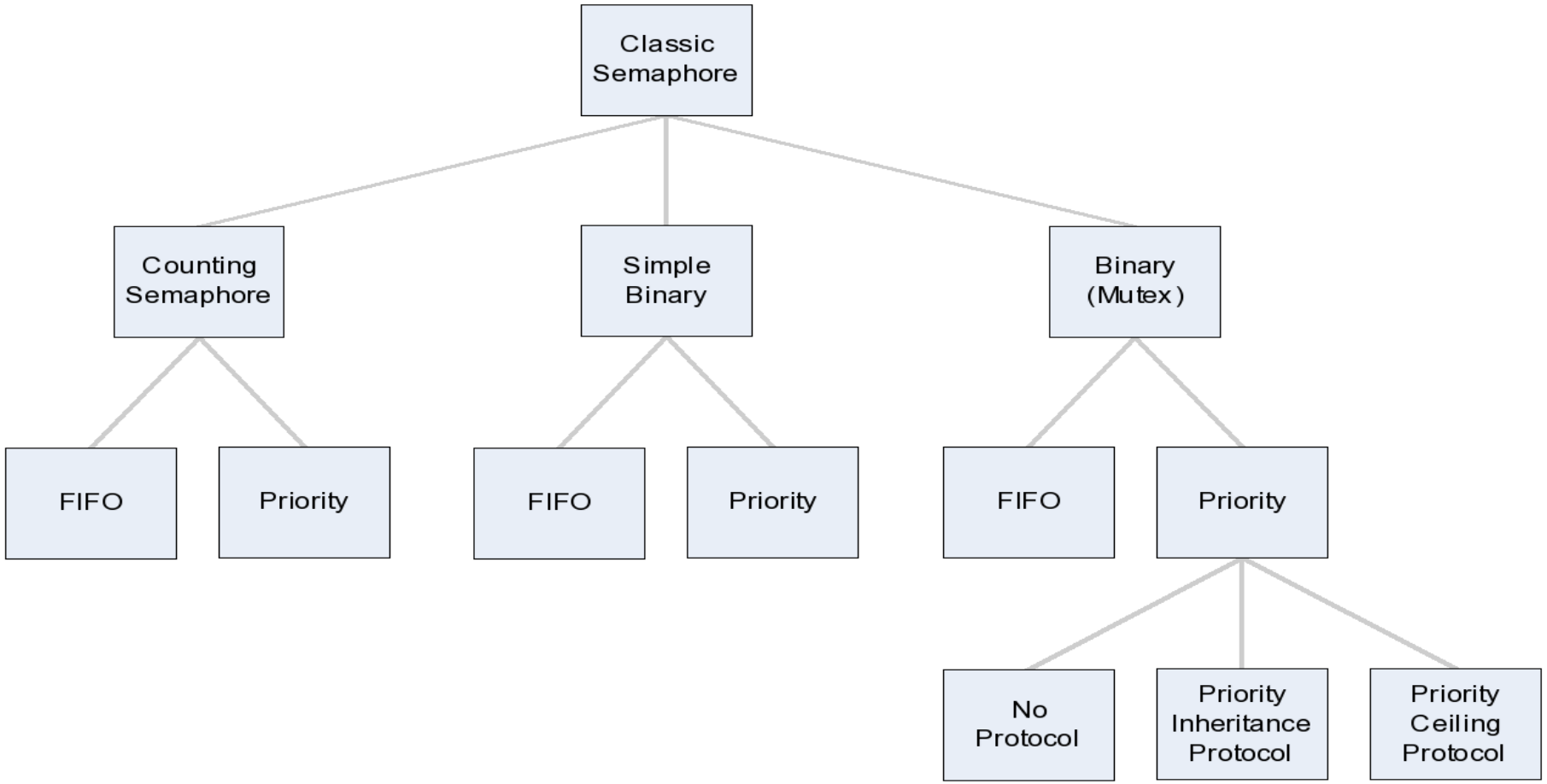
- Właściwości semaforów określane są na etapie tworzenia semaforów,
- Do konfiguracji semaforów wykorzystywane są makra dostępne w systemie RTEMS,

  Makra łączone są przy pomocy sumy bitowej (bitwise OR).
  - RTEMS_FIFO – tworzony semafor zgodnie z algorytmem FIFO (default)
  - RTEMS_PRIORITY – tworzony semafor zgodnie z priorytetem zadania
  - RTEMS_BINARY_SEMAPHORE – semafor binarny (wartości ograniczone do 0, 1)
  - RTEMS_COUNTING_SEMAPHORE – semafor zliczający (default)
  - RTEMS_SIMPLE_BINARY_SEMAPHORE – semafor binarny, niezagnieżdzony, możliwość usunięcia zablokowanych semaforów
  - RTEMS_NO_INHERIT_PRIORITY – semafor bez algorytmem dziedziczenia prior. (default)
  - RTEMS_INHERIT_PRIORITY – semafor z algorytmem dziedziczenia priorytetów
  - RTEMS_PRIORITY_CEILING – semafor z algorytmem pułapu priorytetów
  - RTEMS_NO_PRIORITY_CEILING – semafor bez algorytmu pułapu priorytetów (default)
  - RTEMS_LOCAL – semafor działający na zadaniach lokalnych (default)
  - RTEMS_GLOBAL – semafor działający na zadaniach globalnych

    #define RTEMS_DEFAULT_ATTRIBUTES ....

Atrybuty sterujące semaforami (2)

Zależności pomiędzy atrybutami semaforów.



Utworzenie semafora (1)

rtems_status_code **rtems_semaphore_create**(rtems_name name, uint32_t count, rtems_attribute attribute_set, rtems_task_priority priority_ceiling, rtems_id *id);

- Funkcja tworzy semafor o zadanych parametrach oraz inicjalizuje go wartością podaną w postaci parametru.

Wartość zwracane przez funkcję rtems_semaphore_create:

- RTEMS_SUCCESSFUL – semafor utworzony poprawnie
- RTEMS_INVALID_NAME – błędna nazwa semafora
- RTEMS_INVALID_ADDRESS – wskaźnik ID nie jest ustawiony
- RTEMS_TOO_MANY – zbyt dużo utworzonych semaforów
- RTEMS_NOT_DEFINED – błędna lista argumentów
- RTEMS_INVALID_NUMBER – błędna wartość początkowa dla sem. binarnego
- RTEMS_MP_NOT_CONFIGURED – niedostępny tryb pracy wieloprocesorowej
- RTEMS_TOO_MANY – zbyt dużo obiektów globalnych

Utworzenie semafora (2)

rtems_status_code **rtems_semaphore_create**(rtems_name name, uint32_t count, rtems_attribute attribute_set, rtems_task_priority priority_ceiling, rtems_id *id);

Parametry funkcji tworzącej semafor:

- name – unikalna nazwa semafora,
- count – wartość jaką inicjalizowany jest semafor,
- attribute_set – lista atrybutów tworzonego semafora,
- priority_ceiling – wartość pułapu priorytetu (powinna zostać ustawiona na wartość priorytetu zadania, które korzysta z semafora + 1),
- id – zwracany wskaźnik do semafora.

Utworzenie semafora (3)

rtems_status_code rtems_semaphore_create(rtems_name name, uint32_t count, **rtems_attribute attribute_set**, rtems_task_priority priority_ceiling, rtems_id *id);

Parametry wykorzystywane podczas tworzenia semafora:

- RTEMS_FIFO – zadania wyzwalane kolejką FIFO (default)
- RTEMS_PRIORITY – zadania wyzwalane priorytetami
- RTEMS_BINARY_SEMAPHORE – semafor binarny
- RTEMS_COUNTING_SEMAPHORE – semafor zliczający (default)
- RTEMS_SIMPLE_BINARY_SEMAPHORE – semafor binarny, bez możliwości zagnieżdżania
- RTEMS_NO_INHERIT_PRIORITY – semafor bez dziedziczenia priorytetów (default)
- RTEMS_INHERIT_PRIORITY – semafor dziedziczący priorytety
- RTEMS_PRIORITY_CEILING – użyj pułapu priorytetów (ang. priority ceiling)
- RTEMS_NO_PRIORITY_CEILING – nie używaj pułapu priorytetów (default)
- RTEMS_LOCAL – semafor operujący na zadaniach lokalnych(default)
- RTEMS_GLOBAL – semafor operujący na zadaniach globalnych

#define RTEMS_DEFAULT_ATTRIBUTES ....

Przykład utworzenia semafora

```
   static rtems_id          Semaphore1;
   rtems_status_code        status;
status = rtems_semaphore_create(rtems_build_name('S','e','m','1'), 1, RTEMS_PRIORITY|
                                                    RTEMS_BINARY_SEMAPHORE|RTEMS_INHERIT_PRIORITY|
                                                    RTEMS_NO_PRIORITY_CEILING|RTEMS_LOCAL, 0,
                                                    &Semaphore1);
    if (status != RTEMS_SUCCESSFUL)
        rtems_panic("Can't create printf mutex1:%d", rtems_status_text(status));
```

Funkcja tworzy lokalny semafor binarny z dziedziczeniem priorytetów o nazwie Sem1, zainicjalizowany wartością 1. Zadania oczekujące na semafor wykonywane są zgodnie z ich priorytetami.

Zajęcie semafora

rtems_status_code rtems_semaphore_obtain(rtems_id id, rtems_option option_set,
rtems_interval timeout);

Funkcja zajmuje semafor, jeżeli jest dostępny.

Parametry funkcji zajmującej semafor:

- id – wskaźnik do semafora,
- option_set – RTEMS_WAIT lub RTEMS_NO_WAIT określa jak funkcja ma się zachować, gdy semafora nie da się obecnie zająć. RTEMS_WAIT nakazuje czekać liczbę „taktów” określoną przez parametr timeout. Jeżeli wybrano RTEMS_NO_WAIT wartość semafora jest dekrementowana, a zadanie trafia do kolejki zadań oczekujących na semafor.
- timeout – czas po jakim należy zaprzestać oczekiwania na semafor (0 – nieskończenie długo)

Kody zwracane przez funkcję:

- RTEMS_SUCCESSFUL – operacja zajęcia semafora zakończenia powodzeniem
- RTEMS_UNSATISFIED – semafor niedostępny
- RTEMS_TIMEOUT – upłynął czas oczekiwania na zajęcie semafora
- RTEMS_OBJECT_WAS_DELETED – semafor został usunięty podczas czekania
- RTEMS_INVALID_ID – błędne ID semafora

Usunięcie semafora

rtems_status_code rtems_semaphore_delete(rtems_id id);

Funkcja zwalnia zasoby zajmowane przez semafor. Zadania blokowane przez semafor wstawiane są do kolejki zadań gotowych do wykonania i realizowane zgodnie z priorytetami. Nie można usunąć semafora binarnego, który jest zajęty.

Kody zwracane przez funkcję:

- RTEMS_SUCCESSFUL – operacja zajęcia semafora zakończenia powodzeniem
- RTEMS_INVALID_ID – błędne ID semafora
- RTEMS_UNSATISFIED – semafor niedostępny
- RTEMS_ILLEGAL_ON_REMOTE_OBJECT – nie można usunąć zdalnego obiektu semafora
- RTEMS_RESOURCE_IN_USE – semafor binarny jest używany

Zwolnienie semafora

rtems_status_code rtems_semaphore_release(rtems_id id);

Funkcja zwalnia semafor – licznik semafora podnoszony jest o 1. Jeżeli wartość semafora jest dodatnia zadanie czekające na semafor jest usuwane z kolejki zadań blokowanych i trafia do kolejki zadań gotowych do wykonania.

Parametry funkcji zajmującej semafor:

- id – wskaźnik do semafora,

Kody zwracane przez funkcję:

- RTEMS_SUCCESSFUL – operacja zajęcia semafora zakończenia powodzeniem
- RTEMS_INVALID_ID – błędne ID semafora
- RTEMS_NOT_OWNER_OF_RESOURCE – zadanie próbujące zwolnić semafor nie jest jego właścicielem

Przykład użycia semafora

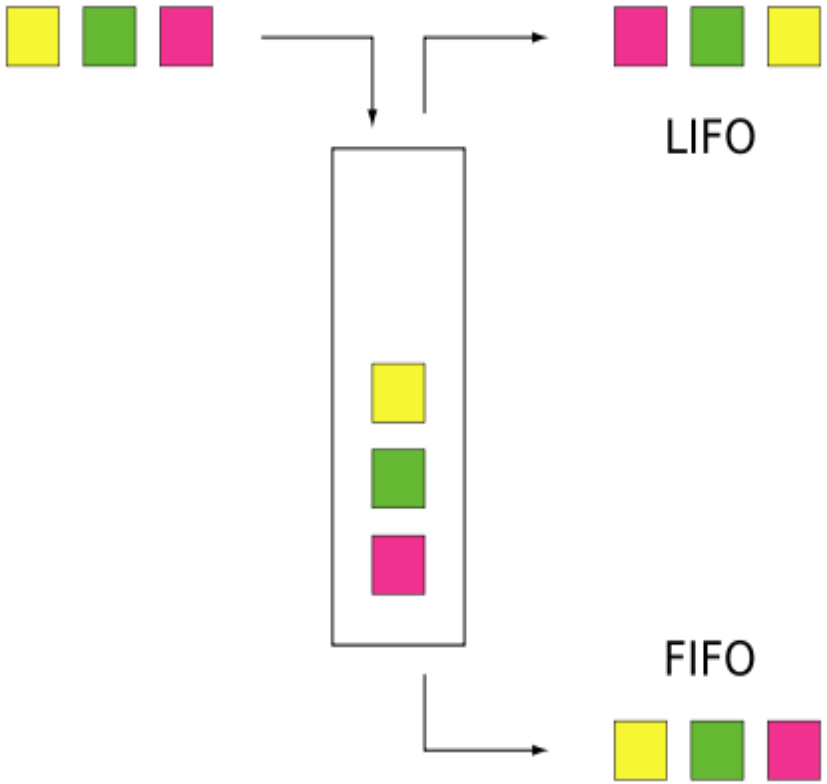
```
  static rtems_id          *Semaphore1, *id;
  Rtems_status_code   status;
status = rtems_semaphore_create(rtems_build_name('S','e','m','1'), 1, RTEMS_PRIORITY|
                                                RTEMS_BINARY_SEMAPHORE|RTEMS_INHERIT_PRIORITY|
                                                RTEMS_NO_PRIORITY_CEILING|RTEMS_LOCAL, 0,
                                                &Semaphore1);
   if (status != RTEMS_SUCCESSFUL)
      rtems_panic("Can't create printf mutex1:%d", rtems_status_text(status));

   rtems_semaphore_obtain(Semaphore1, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

------ sekcja krytyczna, obniżenie semafora do wartości 0

   if (ready==1) printf ("Data: %d\n", data);
   Ready=0;
         status = rtems_semaphore_ident(rtems_build_name('S','e','m','1'), RTEMS_SEARCH_ALL_NODES,
                                                id);
         printf ("Sem1 semaphore ID %d\n", id);

------ koniec sekcji krytycznej, podniesienie semafora do wartości 1

   rtems_semaphore_release(Semaphore1);
```

Kolejki komunikatów

Kolejki komunikatów

Kolejki komunikatów wykorzystywane są do bezpiecznego przesyłania danych między procesami lub procedurami obsługi przerwań oraz do synchronizacji procesów.

Komunikaty przechowywane są w zmiennej długości buforze FIFO (LIFO). Użytkownik może samodzielnie określić rozmiar wiadomości oraz ich zawartość.



Message Manager - atrybuty

Właściwości kolejki komunikatów określane są na etapie tworzenia kolejki.

Do konfiguracji kolejek wykorzystywane są makra dostępne w systemie RTEMS łączone przy pomocy sumy logicznej.

Parametry wykorzystywane podczas tworzenia kolejki:

- RTEMS_FIFO – wykonywanie zadań zgodnie z algorytmem FIFO (default)
- RTEMS_PRIORITY – wykonywanie zadań według priorytetów
- RTEMS_LOCAL – lokalna kolejka komunikatów (default)
- RTEMS_GLOBAL – globalna kolejka komunikatów

Parametry wykorzystywane podczas odczytywania komunikatów z kolejki:

- RTEMS_WAIT – zadanie będzie oczekiwało na komunikat (default)
- RTEMS_NO_WAIT – zadanie nie będzie czekało.

Message Manager

RTEMS udostępnia zestaw funkcji pozwalających na utworzenie i obsługę kolejki komunikatów.

Funkcje operujące na kolejkach komunikatów:

- rtems_message_queue_create – utwórz kolejke
- rtems_message_queue_ident – pobierz ID utworzonej kolejki
- rtems_message_queue_delete – usuń kolejke
- rtems_message_queue_send – wstaw komunikat na koniec kolejki
- rtems_message_queue_urgent – wstaw komunikat na początek kolejki
- rtems_message_queue_broadcast – przesłanie komunikatów rozgłoszeniowych broadcast
- rtems_message_queue_receive – odczytaj komunikat z kolejki
- rtems_message_queue_get_number_pending – pobierz liczbę komunikatów w kolejce
- rtems_message_queue_flush – usuń komunikaty z kolejki

Utworzenie kolejki (1)

```
rtems_status_code rtems_message_queue_create(rtems_name name, uint32_t count,
                                             size_t max_message_size, rtems_attribute
                                             attribute_set, rtems_id *id);
```

Funkcja tworzy kolejkę komunikatów zgodnie z podanymi parametrami. RTEMS tworzy rekord QCB opisujący kolejkę.

Kody zwracane przez funkcję:

- RTEMS_SUCCESSFUL – kolejka utworzona poprawnie,
- RTEMS_INVALID_NAME – nieprawidłowa nazwa kolejki,
- RTEMS_INVALID_ADDRESS – błędna wartość ID,
- RTEMS_INVALID_NUMBER – błędna liczba wiadomości,
- RTEMS_INVALID_SIZE – błędny rozmiar wiadomości,
- RTEMS_TOO_MANY – zbyt dużo utworzonych kolejek,
- RTEMS_UNSATISFIED – brak miejsca na alokacje bufora kolejki,
- RTEMS_MP_NOT_CONFIGURED – system wieloprocesorowy nie dostępny,
- RTEMS_TOO_MANY – zbyt dużo globalnych obiektów.

Utworzenie kolejki (2)

```
rtems_status_code rtems_message_queue_create(rtems_name name, uint32_t count,
                                             size_t max_message_size, rtems_attribute
                                             attribute_set, rtems_id *id);
```

Parametry funkcji tworzącej kolejkę:

- name – nazwa kolejki
- count – liczba komunikatów, które można zapisać w kolejce,
- max_message_size – rozmiar komunikatu,
- attribute_set – atrybuty określające właściwości kolejki,
- id – wskaźnik do utworzonej kolejki.

```
static rtems_id          *Queue1;
rtems_status_code        status;
status = rtems_message_queue_create(rtems_build_name('Q','U','E','U'), 10,
                                    sizeof(dataFrame), RTEMS_DEFAULT_ATTRIBUTES, Queue1);
 if (status != RTEMS_SUCCESSFUL) {
      process_queue_exception();
 }
```

Usunięcie kolejki

rtems_status_code rtems_message_queue_delete(rtems_id *id);

Funkcja usuwa utworzoną kolejkę. Zablokowane zadania oczekujące na dane zostają odblokowane. Zasoby zużywane przez kolejkę są przywracane do systemu RTEMS. Komunikaty przechowywane w kolejce są tracone.

Wartości zwracane przez funkcję:

- RTEMS_SUCCESSFUL – kolejka usunięta poprawnie,
- RTEMS_INVALID_ID – niewłaściwy wskaźnik ID,
- RTEMS_ILLEGAL_ON_REMOTE_OBJECT – nie można usunąć zdalnego obiektu.

Zapisanie danej w kolejce (1)

rtems_status_code rtems_message_queue_send(rtems_id *id, const void *buffer, size_t size);

Funkcja przesyła komunikat o rozmiarze size (bajtów) do kolejki. Komunikat wstawiany jest na końcu kolejki FIFO.

Wartości zwracane przez funkcję:

- RTEMS_SUCCESSFUL – komunikat zapisany w kolejce pomyślnie,
- RTEMS_INVALID_ID – niewłaściwy wskaźnik ID,
- RTEMS_INVALID_SIZE – niewłaściwy rozmiar komunikatu,
- RTEMS_INVALID_ADDRESS – wskaźnik buffer ustawiony niepoprawnie,
- RTEMS_UNSATISFIED – próba zapisania danych poza buforem FIFO,
- RTEMS_TOO_MANY – brak miejsca w kolejce.

Zapisanie danej w kolejce (2)

```
rtems_status_code rtems_message_queue_urgent(rtems_id *id, const void *buffer,
                                             size_t size);
```

Funkcja przesyła komunikat o rozmiarze size (bajtów) do kolejki. Komunikat wstawiany jest na początku kolejki FIFO.

Wartości zwracane przez funkcję:

- RTEMS_SUCCESSFUL – komunikat zapisany w kolejce pomyślnie,
- RTEMS_INVALID_ID – niewłaściwy wskaźnik ID,
- RTEMS_INVALID_SIZE – niewłaściwy rozmiar komunikatu,
- RTEMS_INVALID_ADDRESS – wskaźnik buffer ustawiony niepoprawnie,
- RTEMS_UNSATISFIED – próba zapisania danych poza buforem FIFO,
- RTEMS_TOO_MANY – brak miejsca w kolejce.

Zapisanie komunikatów broadcast w kolejce

```
rtems_status_code rtems_message_queue_broadcast(rtems_id *id, const void *buffer,
                                                size_t size, uint32_t *count);
```

Funkcja przysyła komunikaty z bufora o rozmiarze size (bajtów) do wszystkich zadań oczekujących na dane. Komunikat nie jest kopiowany do kolejki FIFO. Funkcja zwraca wartość liczbową informująca o liczbie zadań odblokowanych do, których dotarła wiadomość.

Wartości zwracane przez funkcję:

- RTEMS_SUCCESSFUL – komunikat broadcast przesłany pomyślnie,
- RTEMS_INVALID_ID – niewłaściwy wskaźnik ID,
- RTEMS_INVALID_ADDRESS – wskaźnik count ustawiony niepoprawnie,
- RTEMS_INVALID_ADDRESS – wskaźnik buffer ustawiony niepoprawnie,
- RTEMS_INVALID_SIZE – błędny rozmiar wiadomości.

Odczyt komunikatu z kolejki

```
rtems_status_code rtems_message_queue_receive(rtems_id *id, void *buffer,
                                              size_t size, rtems_option option_set,
                                              rtems_interval  timeout);
```

Funkcja odczytuje komunikat z kolejki i zapisuje go w buforze (buffer) o rozmiarze size. Jeżeli wywołano funkcję z parametrem RTEMS_WAIT, a w kolejce nie ma danych funkcja czeka na dane. Czas oczekiwania zależy od wartości timeout. Jeżeli użyto parametru RTEMS_NO_WAIT funkcje nie czaka. Parametr size określa rozmiar odczytanych danych.

Wartości zwracane przez funkcję:

- RTEMS_SUCCESSFUL – komunikat odczytany pomyślnie,
- RTEMS_INVALID_ID – niewłaściwy wskaźnik ID,
- RTEMS_INVALID_ADDRESS – wskaźnik count ustawiony niepoprawnie,
- RTEMS_INVALID_ADDRESS – wskaźnik buffer ustawiony niepoprawnie,
- RTEMS_UNSATISFIED – brak wiadomości w kolejce
- RTEMS_TIMEOUT – timeout dla funkcji oczekującej
- RTEMS_OBJECT_WAS_DELETED – kolejka usunięta podczas oczekiwania na wiadomość

Informacje o liczbie komunikatów w kolejce

rtems_status_code rtems_message_queue_get_number_pending(rtems_id *id,uint32_t *count);

Funkcja zwraca liczbę wiadomości (count) w kolejce FIFO.

Wartości zwracane przez funkcję:

- RTEMS_SUCCESSFUL – komunikat odczytany pomyślnie,
- RTEMS_INVALID_ID – niewłaściwy wskaźnik ID,
- RTEMS_INVALID_ADDRESS – wskaźnik count ustawiony niepoprawnie.

Usunięcie danych z kolejki

rtems_status_code rtems_message_queue_flush(rtems_id *id, uint32_t *count);

Funkcja usuwa wszystkie wiadomości z kolejki wskazanej przez wskaźnik ID. Liczba usuniętych wiadomości zwracana jest w zmiennej count.

Wartości zwracane przez funkcję:

- RTEMS_SUCCESSFUL – komunikat odczytany pomyślnie,
- RTEMS_INVALID_ID – niewłaściwy wskaźnik ID,
- RTEMS_INVALID_ADDRESS – wskaźnik count ustawiony niepoprawnie.

Przykład użycia kolejki FIFO (1)

```
typedef struct _FRAME
{
        rtems_unsigned16 frame_start;
        rtems_unsigned8  sender_address;
        rtems_unsigned8  recv_address;
        rtems_unsigned8      cmd;
        rtems_unsigned32 data;
        rtems_unsigned16 frame_end;
}__attribute__((packed)) FRAME, *PFRAME;

rtems_status_code   status;
FRAME               frame, myframe;
rtems_id            queue_id;

status = rtems_message_queue_create(rtems_build_name('Q','U','E','0'), 10, sizeof(FRAME),
RTEMS_DEFAULT_ATTRIBUTES, &queue_id);
if (status != RTEMS_SUCCESSFUL){
        rtems_panic("Can't create task: %s", rtems_status_text(status));
}
```

Przykład użycia kolejki FIFO (2)

```
rtems_unsigned32    size;
status = rtems_message_queue_flush(queue_id, &size);

/* send myframe buffer to queue */
status = rtems_message_queue_send(queue_id, myframe, sizeof(FRAME));



/* receive message from queue, blocking mode */
status = rtems_message_queue_receive(queue_id, frame_buff, &size,
                                     RTEMS_DEFAULT_ATTRIBUTES, RTEMS_NO_TIMEOUT);

size = lenght of received bytes of data
frame_buffer = buffer large enough to house receive portion of data

/* receive message from queue, with timeout in ms*/
status = rtems_message_queue_receive(queue_id, frame_buff, &size,
                            RTEMS_DEFAULT_ATTRIBUTES,
                            CYCLE_TIME/rtems_configuration_get_milliseconds_per_tick());
```

IO Menager
(zarządzanie portami wej./wyj.)

Sterowanie portami wejścia-wyjścia

- Dostęp do portów I/O procesora jest możliwy z wykorzystaniem bezpośrednich odwołań do rejestrów procesora (przy pomocy wskaźnika), jednak taka metoda nie powinna być stosowana,
- Urządzenia peryferyjne powinny wykorzystywać specjalizowane sterowniki (UART, wyświetlacze, LED, LCD, itd...),
- RTEMS udostępnia mechanizm pozwalający na rejestrowanie nowych sterowników.
- Przy pomocy sterownika I/O można w łatwy i bezpieczny sposób odczytywać i sterować wyprowadzeniami procesora oraz urządzeniami peryferyjnymi.

IO Manager

RTEMS udostępnia następujące funkcje służące do implementacji sterowników urządzeń:

- rtems_io_initialize - Initialize a device driver,
- rtems_io_register_driver - Register a device driver,
- rtems_io_unregister_driver - Unregister a device driver,
- rtems_io_register_name - Register a device name,
- rtems_io_lookup_name - Look up a device name,
- rtems_io_open - Open a device,
- rtems_io_close - Close a device,
- rtems_io_read - Read from a device,
- rtems_io_write - Write to a device,
- rtems_io_control - Special device services.