Inter-process communication


uVision5 Books Pane: “MDK-ARM Getting Started” (PDF), CMSIS-RTOS2 (pp26-36)
Keil directory: C:/Keil/ARM/PACK/ARM/CMSIS/5.3.0/CMSIS/RTOS2
(user code templates, examples, documentation)
Interprocess communication

- **Interprocess communication (IPC):** OS provides mechanisms so that processes can pass data.

- **Two types of semantics:**
  - **blocking:** sending process waits for response;
    - time limit might be set in some cases
  - **non-blocking:** sending process continues.
Interprocess communication (IPC) mechanisms

- Semaphores
  - binary
  - counting
- Signals
- Mail boxes
- Queues
- Pipes
CMSIS-RTOS2 inter-thread communication

- **Thread flag** – for thread synchronization
  - Each thread has a pre-allocated 32-bit thread flag object.
  - A thread can wait for its TFs to be set by threads/interrupts.

- **Event flag** – for thread synchronization
  - Similar to thread flags, except dynamically created

- **Semaphores** – control access to common resource
  - Semaphore object contains tokens (“counting” semaphore)
  - Thread can request a token (put to sleep if none available)

- **Mutex** – mutual exclusion locks
  - “lock” a resource to use it, and unlock it when done
  - Kernel suspends threads that need the resource until unlocked

- **Message Queue** (Mail Queues eliminated in RTOS2)
  - Queue is a first-in/first-out (FIFO) structure
  - “Message” is an integer or a pointer to a message frame
  - Suspend thread if “put” to full queue or “get” from empty queue
Interprocess communication styles

- **Shared memory:**
  - processes have some memory in common;
  - cooperate to avoid destroying/missing messages.

- **Message passing:**
  - processes send messages along a communication channel
    ---no common address space.
  - comm. channel may be physical or virtual
Shared memory

- CPUs could be separate processors and/or cores within a processor
- Multiple processes on the same CPU may also share memory

- Shared memory on a bus:
Race condition in shared memory

- Assume a “flag” used to synchronize access to shared memory
  - Flag = 1 when shared item is being used
  - Flag = 0 when shared item is not being used
  - To access the item: CPU must see flag = 0 and write flag = 1

- Problem when two CPUs try to write the same location:
  - CPU 1 reads flag and sees 0.
  - CPU 2 reads flag and sees 0.
  - CPU 1 sets flag to one and writes location.
  - CPU 2 sets flag to one and overwrites location.
Atomic test-and-set

- Problem can be solved with an atomic test-and-set:
  - single bus operation reads memory location, tests it, writes it.
- ARM test-and-set provided by SWP (swap):

```
ADR    r0,SEMAPHORE
LDR    r1,#1
GETFLAG SWP   r1,r1,[r0]
BNZ    GETFLAG
```
Critical regions

- **Critical region**: section of code that cannot be interrupted by another process.

- Examples:
  - writing shared memory;
  - accessing I/O device.
Mutual Exclusion Example

System variables: Lock = 0;

while (1){
    While (!Test-And-Set(&Lock));
    Critical Region
    Lock = 0;
    Remainder
} /* end while (1) */

Process 1

while (1){
    While (!Test-And-Set(&Lock));
    Critical Region
    Lock = 0;
    Remainder
} /* end while (1) */

Process 2
Task and Device Queues

Processes queued for shared device access
Semaphores

- **Semaphore**: OS primitive for controlling access to critical regions.
- Sempaphore can be “binary” or “counting”
- **Protocol:**
  1. Get access to semaphore with $P()$ function.  
     *(Dutch “Proberen” – to test)*
  2. Perform critical region operations.
  3. Release semaphore with $V()$ function.  
     *(Dutch “Verhogen” – to increment)*

- Sempaphore may be “binary” or “counting”
Binary semaphore

- **Semaphore S values**
  - $S=1$ : resource in use
  - $S=0$ : resource not in use

- **Semaphore S actions**
  - `wait(&S)`: test & set (read $S$, set $S=1$)
    use resource if $S$ was read as 0
    wait if $S$ was read as 1
  - `signal(&S)`: write $S=0$ to free up resource
Counting semaphore

- **Semaphore S values**
  - $S = 1$: resource free
  - $S = 0$: resource in use, no others waiting
  - $S < 0$: resource in use, others waiting

- **Semaphore S actions**
  - `wait(&S)`: $S--$, use resource if $S = 0$, o/w wait
  - `signal(&S)`: $S++$, wake up other task if $S < 1$

Also use for access to $N$ copies of a resource – semaphore indicates number of copies free
Example

- **Access critical region**
  
  ```c
  wait(&S); //continue if read S=1, o/w wait
  //execute “critical region”
  signal(&S); //free the resource
  ```

- **Task synchronization**
  
  ```c
  Task1
  signal(&S1)    signal(&S2)
  wait (&S2)     wait(&S1)
  ```

  tasks synchronize at this point
Potential deadlock

- Tasks 1 and 2 each require two resources, R1 and R2, with access controlled by S1 and S2, respectively

Task 1
wait(&S1)
//have R1
wait(&S2)
//wait for R2

Task 2
wait(&S2)
//have R2
wait(&S1)
//wait for R1

DEADLOCK!!
Mutual Exclusion (MUTEX)

- **Binary semaphore**
- **Provide exclusive access to a resource**
  
  ```c
  osMutexId_t m_id; //MUTEX ID
  m_id = osMutexNew(attr); //create MUTEX obj
  
  attr = osMutexAttr_t structure or NULL for default
  
  status = osMutexAcquire(m_id, timeout);
  
  Wait until MUTEX available or until time = “timeout”
  
  timeout = 0 to return immediately
  
  timeout = osWaitForever for infinite wait
  
  “status” = osOK if MUTEX acquired
  
  osErrorTimeout if not acquired within timeout
  
  osErrorResource if not acquired when timeout=0 specified
  
  status = osMutexRelease(m_id); //release the MUTEX
  
  status = osOK if released, osErrorResource if invalid operation (not owner)
  ```

  Timeout arguments
  
  for other objects have same options
Limit access to shared resource to one thread at a time.

Special version of a “semaphore”

\[
\text{osMutexAcquire}(\text{mutex}_id, \text{timeout}) \\
\text{osMutexRelease}(\text{mutex}_id)
\]
CMSIS-RTOS2 Semaphores

- **Counting semaphore**
- **Allow up to t threads to access a resource**
  
  ```c
  osSemaphoreId s_id;       //semaphore ID
  s_id = osSemaphoreNew(max_tokens, init_tokens, attr);
  // Create s1; set max and initial #tokens
  // attr osSemaphoreAttr_t structure or NULL for defaults
  
  status = osSemaphoreAcquire(s_id, timeout);
  // Wait until token available or timeout
  status = osOK if token obtained (#tokens decremented)
  osErrorTimeout if token not obtained before timeout
  osErrorResource if token not obtained and timeout=0
  
  status = osSemaphoreRelease(s_id);
  // Release token
  status = osOK if token released (#tokens incremented)
  osErrorResource if max token count reached
  osErrorParameter if s_id invalid
  ```
Permit fixed number of threads/ISRs to access a pool of shared resources.

Initialize with max# of “tokens”.

- `osSemaphoreAcquire(sem_id,timeout)`
- `osSemaphoreRelease(sem_id)`
- `osSemaphoreGetCount(sem_id)`
CMSIS-RTOS semaphore example

```c
osSemaphoreId_t sid_Thread_Semaphore; // semaphore id

// Main thread: Create the semaphore
sid_Thread_Semaphore = osSemaphoreNew(2, 2, NULL); // init with 2 tokens
if (!sid_Thread_Semaphore) { // Semaphore object not created, handle failure }

// Application thread: Acquire semaphore - perform task - release semaphore
osStatus_t val;
val = osSemaphoreWait (sid_Thread_Semaphore, 10); // wait up to 10 ticks
switch (val) {
    case osOK: // Semaphore acquired
        // Use protected code here...
        osSemaphoreRelease (sid_Thread_Semaphore); // Return token back to a semaphore
        break;
    case osErrorTimeout: break; // Not acquired within timeout
    case osErrorResource: break; // Not acquired and timeout=0 ("just checking")
    default: break; // Other errors
}
```
POSIX semaphores

- POSIX supports counting semaphores with \_POSIX\_SEMAPHORES option.
  - Semaphore with N resources will not block until N processes hold the semaphore.

- Semaphores are given name:
  - Example: /sem1

- P() is sem_wait()
- V() is sem_post()
int i, oflags;
sem_t *my_semaphore; //descriptor for sem.

//create the semaphore
my_semaphore = sem_open("/sem1", oflags);
/* do useful work here */

//destroy the semaphore if no longer needed
i = sem_close(my_semaphore);
Semaphore example (2)

int i;

i = sem_wait(my_semaphore); // P()
// wait for semaphore, block if not free
// now do useful work
i = sem_post(my_semaphore); // V()

// test without blocking
i = sem_trywait(my_semaphore);
Signals

- Originally, a Unix mechanism for simple communication between processes.
- Analogous to an interrupt---forces execution of a process at a given location.
  - But a signal is generated by one process with a function call.
- No data---can only pass type of signal.
CMSIS-RTOS2 Thread Flags

- Thread flags not “created” – a 32-bit word with 31 thread flags; exists automatically within each thread.

  0 \( \text{TF}_{30} \) \text{... \text{TF}}_{2} \text{ TF}_{1} \text{ TF}_{0} \)

- One thread sets TFs in another thread (addressed by its thread ID)
- \text{osThreadFlagsSet}(\text{tid, flags}) \quad \text{– set TFs of thread tid}
  - flags = int32_t; each “1” bit in “flags” sets the corresponding TF
  - Example: flags=0x8002 => set/clear TF #15 and TF #1

- \text{osThreadFlagsWait}(\text{flags, option, timeout})
  - Wait for TFs corresponding to “1” bits in “flags” to be set
  - Option = osFlagsWaitAny or osFlagsWaitAll = wait for any or all of the flags
  - Timeout = 0 (check and return), osWaitForever, or time T
  - Return 32-bit value of flags (and then clear them)
    - osFlagsErrorTimeout if TFs are set before timeout T
    - osFlagsEventTimeout if no flag before timeout

- \text{osThreadFlagsClear}(\text{tid, flags}) \quad \text{– clear TFs of thread, return current flags set}
- \text{osThreadFlagsGet}() \quad \text{– return flags currently set in this thread}
CMSIS-RTOS thread flags example

//Thread 1
void ledOn (void constant *argument) {
    for (;;) {
        LED_On(0);
        osThreadFlagsSet(tid_ledOff, 0x0001);  //signal ledOff thread
        osDelay(2000);
    }
}

// Thread 2
void ledOff (void constant *argument) {
    for (;;) {
        // wait for signal from ledOn thread
        osThreadFlagsWait(0x0001, osFlagsWaitAny, osWaitForever);
        osDelay(500);
        LED_Off(0);
    }
}
Thread Flag Example – Thread3 must wait for signals from both Thread1 and Thread2

#include "cmsis_os2.h"

osThreadId_t tid1; // three threads
osThreadId_t tid2;
osThreadId_t tid3;

void thread1 (void *argument) {
    while (1) {
        osThreadFlagsSet(tid3, 0x0001); /* signal thread 3 */
        ....
    }
}

void thread2 (void *argument) {
    while (1) {
        osThreadFlagsSet(tid3, 0x0002); /* signal thread 3 */
        ....
    }
}

void thread3 (void *argument) {
    uint32_t flags;
    while (1) {
        // wait for signals from both thread1 and thread2
        flags = osThreadFlagsWait(0x0003, osFlagsWaitAll, osWaitForever);
        ... // continue processing
CMSIS-RTOS2 Event Flags

- Each “signal” has up to 31 “event flags” (bits 30-0 of the signal word)
- Similar to Thread Flags, but Event Flags do not “belong” to any thread
  - Wait (in BLOCKED state) for an event flag to be set
  - Set/Clear one or more event flags

```c
osEventFlagsId_t evt_id;
evt_id = osEventFlagsNew(*attr) – create & initialize event flags
```
- NULL argument for default values (or pointer to osEventFlagsAttr_t structure)
- Return event flags id (evt_id)

```c
osEventFlagsSet(evt_id, flags)    – set EFs in evt_id
osEventFlagsClear(evt_id, flags) – clear EFs of evt_id
```
- flags = int32_t; each “1” bit in “flags” sets/clears the corresponding EF
- Return int32_t = flags after executing the set/clear (or error code)

```c
osEventFlagsWait(evt_id, flags, options, timeout)
```
- Wait for EFs corresponding to “1” bits in “flags” to be set, or until timeout
- Options – osFlagsWaitAny or osFlagsWaitAll (any or all of the indicated flags)
- Return current event flags or error code
  - osFlagsErrorTImeout if awaited flags not set before timeout
  - osFlagsErrorResource if evt_id not ready to be used
Event flags example

```c
osEventFlagsId_t led_flag;

void main_app (void constant *argument) {
    led_flag = osEventFlagsNew(NULL);  //create the event flag
}

void ledOn (void constant *argument) {
    for (;;) {
        LED_On(0);
        osEventFlagsSet(led_flag, 0x0001);    //signal ledOff thread
        osDelay(2000);
    }
}

void ledOff (void constant *argument) {
    for (;;) { // wait for signal from ledOn thread
        osEventFlagsWait(led_flag, 0x0001, osFlagsWaitAny, osWaitForever);
        osDelay(500);
        LED_Off(0);
    }
}
```
POSIX signals

- Must declare a signal handler for the process using `sigaction()`.
  - what to do when signal received
  - handler is called when signal is received
  
  ```c
  retval=sigaction(SIGUSR1,&act,&oldact);
  ```

- Send signal with `sigqueue()`:
  ```c
  sigqueue(destpid,SIGRTMAX-1,sval);
  ```
POSIX signal types *(partial list)*

- **SIGABRT**: abort process
- **SIGTERM**: terminate process
- **SIGFPE**: floating point exception
- **SIGILL**: illegal instruction
- **SIGKILL**: unavoidable process termination
- **SIGALRM**: real-time clock expired
- **SIGUSR1, SIGUSR2**: user defined
Message passing

- Message passing on a network:
Message passing via mailboxes

- Mailbox = message buffer between two processes (FIFO)

Use semaphore to lock buffer during read/write
CMSIS-RTOS2
Message queues

“Message” = information to be sent

- `osMessageQueueId q_id;`  // ID of queue object
- `q_id = osMessageQueueNew(msg-count, msg-size, attr);`
  - Create and initialize a message queue, return queue ID
  - Specify: max #msgs, max msg size, attributes (or NULL for defaults)
- `status = osMessageQueuePut(q_id, msg-ptr, msg-priority, timeout);`
  - Add message to queue; wait for “timeout” if queue full
  - `msg-ptr =` pointer to message data structure
  - Status = `osOK`: msg was put into the queue
    - `osErrorResource`: not enough space for msg
    - `osErrorTimeout`: no memory available at timeout
- `status = osMessageQueueGet(q_id, msg-ptr, msg-priority, timeout);`
  - Get msg from queue, put it in *msg-ptr and put priority in *msg-priority;
  - Wait for “timeout” if no message
  - Status = `osOK`: message was retrieved from the queue
    - `osErrorResource`: no message available and timeout=0
    - `osErrorTimeout`: no message available before timeout
osMessageQueuePut(mq_id, *msg_ptr, msg_prio, timeout)
osMessageQueueGet(mq_id, *msg_ptr, *msg_prio, timeout)

osMessageQueueGetCapacity(mq_id) - max #msgs in the queue
osMessageQueueGetMsgSize(mq_id) - max msg size in memory pool
osMessageQueueGetCount(mq_id) - # queued msgs in the queue
osMessageQueueGetSpace(mq_id) - # available slots in the queue
osMessageQueueReset(mq_id) - reset to empty
CMSIS-RTOS message queue example

// “Message” will be a 32-bit integer
osMessageQueueId_t mid_MyQueue; // message queue id

// Thread 1 code
uint32_t n;

n = something;

osMessageQueuePut(mid_MyQueue, &n, 0, osWaitForever); //send n as the message

// Thread 2 code
osStatus_t status; //function call status
uint32_t msg; //variable for received “message”

status = osMessageQueueGet(qid_MyQueue, &msg, 0, 0); //return immediately if no message

if (status == osOK) //was there a message?
{
    //process its data
}

....

// Main creates threads, message queues, etc.

int main(void)
{
    qid_MyQueue = osMessageQueueNew(16, sizeof(uint32_t), NULL);
/* Message Queue creation & usage example */

// message object data type
typedef struct {
    uint8_t Buf[32];
    uint8_t Idx;
} MSGQUEUE_OBJ_t;

// message queue id
osMessageQueueId_t mid_MsgQ;

// thread creates a message queue for 12 messages
int Init_MsgQueue (void) {
    mid_MsgQ = osMessageQueueNew(12, sizeof(MSGQUEUE_OBJ_t), NULL);
    ....
}

Continued on next slide
/* Message Queue Example Continued */

void Thread1 (void *argument) { // this thread sends data to Thread2
    MSGQUEUE_OBJ_t msg;
    while (1) {
        // Insert thread code here...
        msg.Buf[0] = 0x55; // data to send
        msg.Idx = 0; // index of data in Buf[]
        osMessageQueuePut (mid_MsgQ, &msg, 0, NULL); // send the message
        osThreadYield (); // suspend thread
    }
}

void Thread2 (void *argument) { // This thread receives data from Thread1
    MSGQUEUE_OBJ_t msg;
    osStatus_t status;
    while (1) {
        // Insert thread code here...
        status = osMessageQueueGet (mid_MsgQ, &msg, NULL, NULL); // wait for message
        if (status == osOK) {
            // process data in msg.Buf[msg.Idx]
        }
    }
}
CMSIS-RTOS mail queues (eliminated in RTOS2)

- `osMailQId q_id;` // ID of mail queue object
- `osMailQDef (name, queue_size, type);` // Macro: mail queue name, # entries, mail type
- `q_id = osMailCreate(osMailQ(name), NULL);`
  - Create and initialize a message queue, return queue ID
- `mptr = osMailAlloc(q_id, timeout);` (osMailCAlloc() – allocate and clear memory)
  - Allocate a memory block in the queue that can be filled with mail info
  - “mptr” = pointer to the memory block (NULL if no memory can be obtained)
  - Wait, with timeout, if necessary for a mail slot to become available
- `status = osMailFree(q_id, mptr);` - free allocated memory

- `status = osMailPut(q_id, mptr);`
  - Add mail (pointed to by mptr) to queue; wait for “timeout” if queue full
  - Status = osOK : mail was put into the queue
    - = osErrorValue : mail was not allocated as a memory slot
- `status = osMailGet(q_id, timeout);`
  - Get mail from queue; wait for “timeout” if no mail available
  - Status = osOK : no mail available and timeout=0
    - = osEventTimeout : no mail available before timeout
    - = osEventMail : mail received, pointer = value.p
MicroC/OS-II Mailboxes

- `OSMboxCreate(msg)`
  - create mail box & insert initial msg
- `OSMboxPost(box, msg)`
  - add msg to box
- `OSMboxAccept(box)`
  - get msg if there, o/w continue
- `OSMboxPend(box, timeout)`
  - get msg if there, o/w wait up to timeout
- `OSMboxQuery(box,&data)`
  - return information about the mailbox
Example: Mentor Graphics “Nucleus” POSIX Kernel

- Kernel
  - Extensions and APIs
    - MMU
    - DDL
    - IPC
    - C++
    - POSIX
    - microTRON
  - Kernel Objects
    - Mailboxes
    - Queues
    - Pipes
    - Semaphores
    - Memory Services
    - Signals
    - Tasks
    - Timers
    - Events
    - Zero Copy Buffers
  - Hardware Specific Components
    - Timer Management
    - Interrupt Handling
    - Thread Management

© Mentor Graphics
Six tasks/five “functions”

- a system timer
- a task that queues messages
- a task that retrieves queued messages
- two instances of a task that uses a resource for 100 “ticks”
- a task that waits for event signals
Demo program: Application structures

#include "nucleus.h" /* OS function def's */

/* Define Nucleus objects */
NU_TASK Task_0;
NU_TASK Task_1;
NU_TASK Task_2;
NU_TASK Task_3;
NU_TASK Task_4;
NU_TASK Task_5;
NU_QUEUE Queue_0;
NU_SEMAPHORE Sempahore_0;
NU_EVENT_GROUP Event_Group_0;
NU_MEMORY_POOL System_Memory;
Demo program: Global variables

/* Define demo global variables */
UNSIGNED Task_Time;
UNSIGNED Task_1_messages_sent;
UNSIGNED Task_2_messages_received;
UNSIGNED Task_2_invalid_messages;
NU_TASK *Who_has_the_resource;
UNSIGNED Event_Detections;
Nucleus POSIX process/task creation
(done by startup procedure)

/* Create a system memory pool to allocate to tasks */
status = NU_Create_Memory_Pool(
    &System_Memory,         //pointer to sys memory
    "SYSMEM",               //name
    first_available_memory, //address of 1st location
    SYSTEM_MEMORY_SIZE,     //size of allocated memory
    50,                     //minimum allocation
    NU_FIFO                 //if memory not available,
);                        //resume tasks in FIFO order
Nucleus POSIX process/task creation (done by startup procedure)

/* Create task 0 */
//allocates memory for task stack from memory pool
NU_Allocate_Memory(&System_Memory, &pointer,
    TASK_STACK_SIZE, NU_NO_SUSPEND);

//create task activation record
status = NU_Create_Task(&Task_0, "TASK 0", task_0, 0,
    NU_NULL, pointer, TASK_STACK_SIZE, 1, 20,
    NU_PREEMPT, NU_START);
Nucleus POSIX process/task creation (done by startup procedure)

/* Create task 1 - Queue sending task*/
NU_Allocate_Memory(&System_Memory, &pointer, TASK_STACK_SIZE, NU_NO_SUSPEND);

status = NU_Create_Task(&Task_2, "TASK 2", task_2, 0, NU_NULL, pointer, TASK_STACK_SIZE, 10, 5, NU_PREEMPT, NU_START);

/* repeat for tasks 2-5 */
void task_0( )
{
    Task_Time = 0;
    while (1) {
        NU_Sleep (100); /*suspend for 100 ticks */

        Task_Time++;    /* increment time */

        /* set event flag to lift suspension of Task 5 */
        status = NU_Set_Events(&Event_Group_0,1,NU_OR);
    }
}
Demo program: Queue-sending task

```c
void task_1( )
{
    Send_Message = 0;
    while (1) {
        /* queue a message */
        /* suspend if queue full or time slice */
        status = ND_Send_To_Queue(&Queue_0,
                                  &Send_Message, 1, NU_SUSPEND);
        Send_Message++;
    }
}
```
void task_2( )
{
    message_expected = 0;
    while (1) {
        /* retrieve a message */
        /* suspend if queue empty or time slice */
        status = ND_Receive_From_Queue(&Queue_0,
                                        &Receive_Message, 1, &received_size,
                                        NU_SUSPEND);
        message_expected++;
    }
}
Demo program: Use resource task
(two instances in the demo)

/* two tasks compete for use of a resource */
void tasks_3_and_4( )
{
    while (1) {
        /* set semaphore to lock resource */
        status = ND_Obtain_Semaphore(&Semaphore_0,
            NU_SUSPEND);

        if (status == NU_SUCCESS) {
            Who_has_resource = ND_Current_Task_Pointer();
            /* hold resource 100 ticks to suspend other task */
            NU_Sleep (100);
            NU_Release_Semaphore (&Semaphore_0);
        }
    }
}
void task_5( )
{
    event_detections = 0;
    while (1) {
        /* wait for event and consume it */
        status = ND_Retrieve_Events(&Event_Group_0, 1,
                                       NU_OR_CONSUME, &event_group, NU_SUSPEND);
        if (status == NU_SUCCESS) {
            Event_Detections++;
        }
    }
}