TOPPERS

Introductory implementation seminar

(JSP-1.4|AKI-H8/3069F | instant noodle timer session)

DAY 2

TOPPERS Project

Educational Working Group

About this document

- **Document Usage Conditions**

1. Copyrights

2. For opinions, proposals, and questions concerning this document, send it by e-mail to the TOPPERS Project secretariat.

This document is subject to change, for content improvement and otherwise, without prior notification.

3.

Seminar schedule

Day 1

1. Checking the development tool
   0.5 hr
2. What is embedded?
   Revising the introductory SESSAM seminar 1.5 hrs
3. Confirming operation of non-OS teaching material 1 hr
4. What is Real-Time OS? 1.5 hrs
   RTOS conditional transfer, confirmation by utilizing HW I/F education material
5. Construction of instant noodle timer
   Constructing the timer 1.5 hrs

Day 2

1. Implementing the instant noodle timer
   Implementing the timer 1.5 hrs
2. Review 0.5 hr
   Evaluation of construction and application
3. Synchronization and communication
   1 hr
   Correspondence between tasks
   Synchronizing and exclusive control
4. Reconstruction using eventflags
   1.5 hrs
5. Summary 1 hr

Implementing the timer

1. Implementing the instant noodle timer
2. Review
3. Synchronization and communication
4. Reconstruction using eventflags
5. Summary

Let us create a RTOS version instant noodle timer.

In this development tool, the use of debugger is not possible so use the system log display function for debugging.

Basic knowledge for implementing the JSP kernel 1

Steps to initializing the system and initialization program

- To start the kernel, minimum initialization to target is needed before calling kernel_start
- CPU is locked while the execution is conducted
- For each target, the JSP kernel readies a startup module for execution
- For details, see manual contained in jsp/doc directory

A target dependent source exists under the config directory.

There is a naming rule for this directory.

<processor name>-<development tool name>

If the <development tool name> is omitted, GNU-GCC development tool name will be used.

The directory below will indicate the <system name>. Commonly, the target board name is used for the system name.

The directory structure for AKI-H8/3069F environment is as follows:

config/h8/akih8_3069f

Normally, the initializing program at power-ON is an assembler language program called *start.S* located under <CPU name>-<development tool name>. At the end of this program, call up for _kernel_start is set.

kernel_start() is described in the C language program *startup.c* located in the kernel directory. *startup.c* is a program which processes initialization and exiting for TOPPERS/JSP kernel.

Basic knowledge for implementing the JSP kernel 2

Confirming the configurator

- With RTOS, memory resource used by the kernel object is obtained statically and/or dynamically
- TOPPERS/JSP kernel determines memory resource when compiling and obtains it statically
- A configurator is a program that automatically sets up the kernel object statically

There are two ways where the kernel object obtains memory after the power is turned ON; one type is dynamically and the other is determined at compile/link time. Most RTOS secures memory dynamically. The TOPPERS full set kernel secures memory dynamically as well. Because there is no need for useless managerial areas when securing memory statically, the total memory size is smaller. The TOPPERS/JSP kernel uses a automatic assignment program called a configurator to secure memory statically.

The configurator creates a source file and include file which defines the memory area from the configuration file (.cfg). [Contents of the instant noodle timer is noted in section 4 of day 2 “Task communication #2”]

The configurator is executed by the cfg command under the cfg directory. Since this program is not CPU dependent, it can be used with any CPU. It is called by the cfg command at the beginning of build, so you can check it in the build log.

JSP kernel timer implementation 1

Possible to execute at 250ms without main routine

- With the OS-less implementation, two processes are forcibly executed every 250ms by main()
- Utilizing the service call tslp_tsk, a task can create a process that executes every 250ms within itself
- Command within the *for* loop of entry_task and timer_task is executed every 250ms without being externally called

With real time operating system, the two processes are executed as tasks. Unlike the non-OS version where execution is called by the main routine, the two processes are executed on separate processors. This means that entry_task and timer_task acts as separate main routines.

If the contents of the switch_process program is set into the *for* loop string of entry_task, and contents of the timer_process program is set into the *for* loop string of timer_task, then execution is possible without a main routine.

JSP kernel timer implementation 2

Communication between the two tasks

- Communication notifying timer condition from ENTRY_TASK to TIMER_TASK is necessary
- What kind of information is communicated?
  1. Timer start up notification
  2. Time extension notification
  3. Timer stop notification
- When notified, three condition transition occurs with TIMER_TASK

  timer start-up, timeout, timer exit

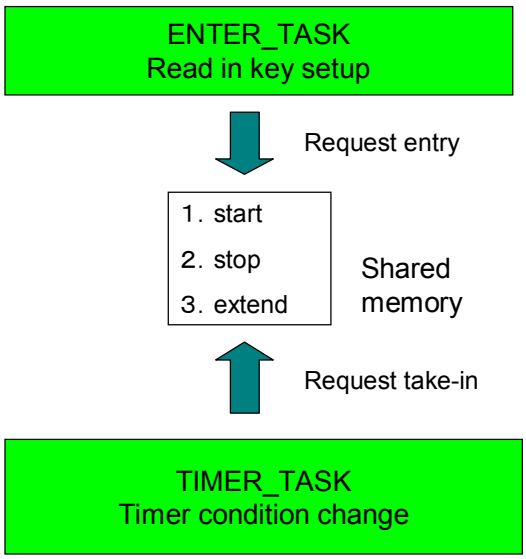
Memory area where multiple tasks share data is called shared memory. Communication can be established by one task writing data to this area and the other task reading in the data from this area.

Create the program so that the data read in by TIMER_TASK recognizes this as an event and relays the state of the timer.

JSP kernel timer implementation 3

Using shared memory

- Information is communicated through shared memory
- Here, one side writes and the other side enters the information



TOPPERS

JSP kernel timer implementation 4

Data communication using shared memory

- Using shared memory (external unsigned char type variable), three conditions are communicated
- ENTRY_TASK writes the communicated information and TIMER_TASK reads the communicated information

JPS kernel timer implementation 5

Using TOPPERS system log

- Using system log function, logs can be displayed to the console as was done with timer1.c source
- exinf value will be displayed on log at ENTRY_TASK startup

  syslog_1(LOG_NOTICE, “Sample entry task starts (exinf=%d).”, exinf)
- Within the log information there is a level of importance setting. LOG INFO is not set to display at default. Changing the display level can be done by the vmsk_log function

Refer to jsp/include/syslog.h for importance level

The log information importance is set by the syslog_n function’s first parameter. Since display is decided by comparing the aforementioned value and logmask value, it becomes possible to switch between displaying/concealing unnecessary logs. With TOPPERS/JSP kernel Release1.4, switching can be done by setting the following function:

```
ER syslog_setmask(UINT logmask, UINT lowmask);
```

```
/*  define log information rank (syslog.h) */
#define LOG_EMERG       0       /* shutdown error  */
#define LOG_ALERT       1
#define LOG_CRIT        2
#define LOG_ERR         3       /* system error */
#define LOG_WARNING     4       /* warning message */
#define LOG_NOTICE      5
#define LOG_INFO        6
#define LOG_DEBUG       7       /* debugging message */
```

Review

1. Implementing the instant noodle timer
2. **Review**
3. Synchronization and communication
4. Reconstruction using eventflags
5. Summary

Let us review the finished program.

There are several techniques to a balanced implementation.

First, about module division

・optimal module size

approximately half page (30 lines)

simplify implementation

・clearly defined system

equality system (left-right balanced, top to bottom is logic-physics)

the system can be interpreted by module relation

・minimize repetition

do not set same function to separate modules

・conceal information

conceal data used by module

Review

Presenting your program

- There is no "correct" program
- Quality of design can be examined

Module division

- optimize size, clarify the system, minimize repetition, and conceal information

Module cohesion

- higher the cohesion, the better the maintenance

Module coupling

- weaker coupling makes better maintenance

TOPPERS

- Module cohesion

Listed in descending order of cohesion level

1. Functional cohesion - module with a singular function, information, and duty
2. Sequential cohesion - parts of a module grouped because the output from one part is the input to another part
3. Communicational cohesion - parts of a module grouped because they operate on the same data
4. Procedural cohesion - parts of a module grouped together which follows a certain sequence of execution
5. Temporal cohesion - parts of a module grouped temporarily together when processing
6. Logical cohesion - parts of a module grouped based on slight relation and is useful when externally used
7. Coincidental cohesion - parts of a module grouped arbitrarily; the parts have no significant relationship

- Module coupling

Listed in ascending order of coupling level

1. Data coupling – singular data without structure
2. Data-structured coupling - modules share a composite data structure
3. Bundling coupling - modules structured by multiple fields
4. Control coupling - one module controls the logic of another
5. External coupling - modules share an externally imposed data and/or flags
6. Common coupling - modules share the same global data
7. Content coupling - one module modifies or relies on the internal workings of another module (assembler)

Review and test

Points to be noted

- The best order of work for reviewing, debugging, and testing depends on the development goal, environment, and developers' skill
- There are instances where programs are made specifically for reviewing, debugging, and testing
- Clarify the code by fine tuning the work, using code organization tools and examine the content by setting grammar/spell checker and compiler warning level to maximum prior to reviewing
- Making full use of techniques is a means to achieving the development goal

There are many expensive static and dynamic program tools on the market.

UNIX command astyle for assembling C, C++, and Java language codes, and UNIX command splint for C language grammar checker are some of the easy to use tools.

Synchronization and communication

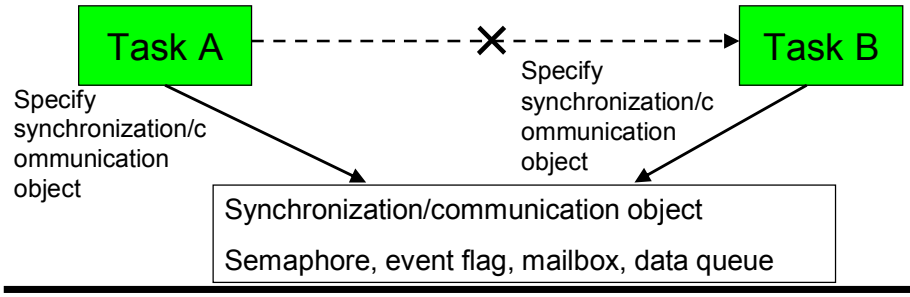
1. Implementing the instant noodle timer
2. Review
3. **Synchronization and communication**
4. Reconstruction using eventflags
5. Summary

Let us study about task synchronization and communication supplied by external objects. Basic synchronization and communication functions are semaphore, eventflag, mailbox, and data queue.

Concept of synchronization/communication between tasks

A task is a virtualized processor

- Synchronization/communication between tasks is the virtualization of synchronization/communication between processors
- Synchronization/communicate without specifying target task by utilizing communication/synchronization object



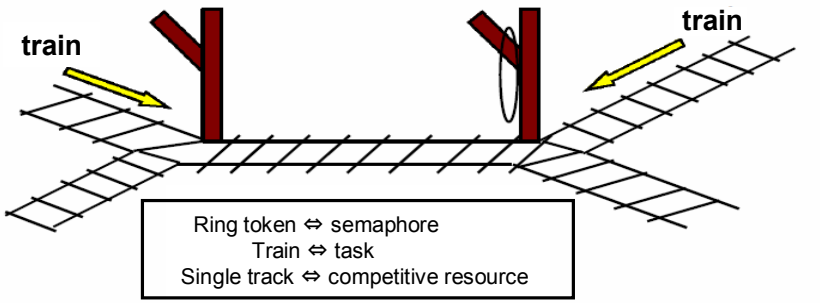
TOPPERS

Semaphore 1

Exclusive control provided by operating system

- Etymology

  A system utilized on single track railway which gave information about the state of the line ahead. Only trains which possessed the ring token could enter into the single track section



TOPPERS

Semaphore, by indicating availability and number of unused resources by counter, is an object that performs exclusive control and/or synchronization when using the resource. When releasing a resource, the semaphore counter is incremented by 1, and decremented by 1 when acquiring a resource.

Semaphore is composed by the counter and a task waiting to obtain the resource wait queue. When the resource count becomes 0, because there is no resource available, a task attempting to acquire a resource is placed in the wait queue until a resource is available.

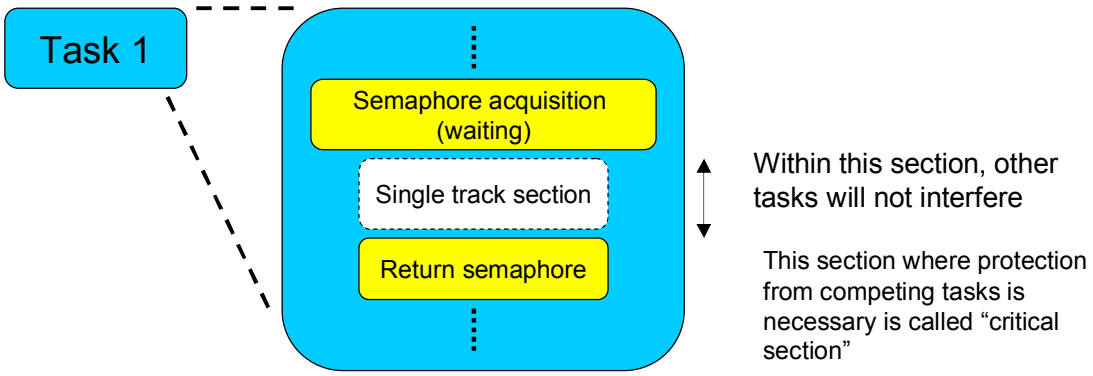
To avoid the case where too many resources are returned to the semaphore, it is possible to set the maximum resource count to a semaphore. Initial semaphore resource count can be set as well. When resources returned exceeds the maximum resource count, an error will be returned.

Semaphore ID is used to identify separate semaphores.

Semaphore 2

Usage

- Correspond one resource to one semaphore



・necessary sections can be protected efficiently
・conservation of source code is improved

TOPPERS

Semaphore service call specification for μITRON4.0 is as follows:

1. Create semaphore

CRE_SEM(ID semid, {ATR sematr, UINT isemcnt, UINT maxsem});

ID semid	semaphore ID number
ATR sematr	semaphore attribute (TA_FIFO || TA_TPRI)
UINT isemcnt	initial semaphore resource count
UNIT maxsem	maximum semaphore resource count

2. Release semaphore resource

ER ercd = sig_sem(ID semid); ER ercd = iseg_sem(ID semid);

ID semid	semaphore ID number of which resource is released

3. Acquire semaphore resource

ER ercd = wai_sem(ID semid); ER ercd = pol_sem(ID semid);

ER ercd = twai_sem(ID semid, TMO tmout);

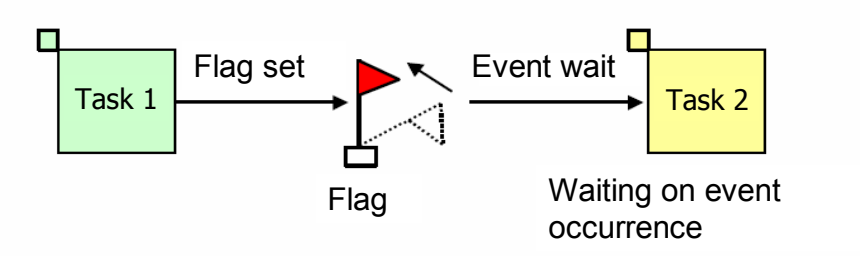
ID semid	semaphore ID number of which resource is acquired
TMO tmout	specified timeout（twai_sem only）

Eventflag 1

How does it work?

- Notifies occurrence of an event to tasks

  Corresponds event and flag



- It is possible to set AND/OR waiting conditions to multiple bit flags

An eventflag is a synchronization object between tasks where events are represented by individual bit flags.

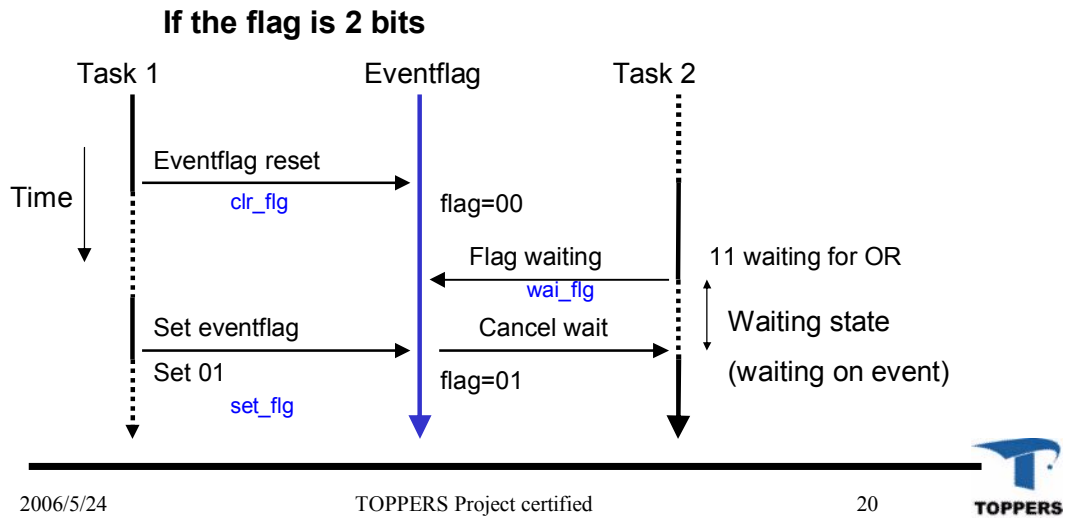
An eventflag has an associated bit pattern expressing the state of the event and a wait queue for tasks waiting on the events. Tasks sending events occurrences are able to set and clear specified bits when necessary. Tasks waiting for events to occur will wait until all specified bit patterns of the eventflag bit is set. Tasks waiting for eventflags are placed in the eventflag wait queue.

Eventflag functions includes creating and deleting eventflags, setting and clearing eventflags, wait for eventflags, and referencing the state of eventflags.

Eventflag 2

Reference state of eventflag

- Notify existence of event by flag bit pattern



Eventflag function service call specifications for μITRON4.0 is as follows:

1. Create eventflag

CRE_FLG(ID flgid, {ATR flgatr, FLGPTN iflgptn});

ID flgid	eventflag ID number
ATR flgatr	eventflag attribute ((TA_TFIFO||TA_PRI) | (TA_WSGL||TA_WMUL)| [TA_CLR])
FLGPRN ifigptn	initial value of eventflag bit pattern

2. Set eventflag

ER ercd = set_flg(ID flgid, FLGPTN setptn);

ER ercd = iset_flg(ID flgid, FLGPTN setptn);

3. Clear eventflag

ER ercd = clr_flg(ID flgid, FLGPTN clrptn);

4. Waiting for eventflag

ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);

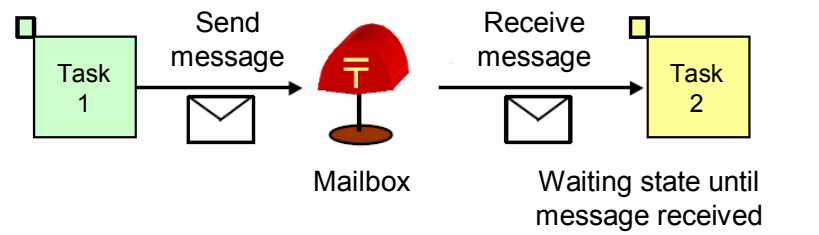
ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);

ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);

Mailbox 1

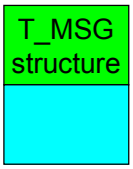
How does it work?

- Send and receive messages (synchronize and communicate simultaneously)



- Data structure of mail

Data load of one message



Header space for the OS

User data space (OS does not intervene)

Define data structure of mail for each application use

```
struct MAIL_tag {
        T_MSG header;
        int data;
}
```


TOPPERS

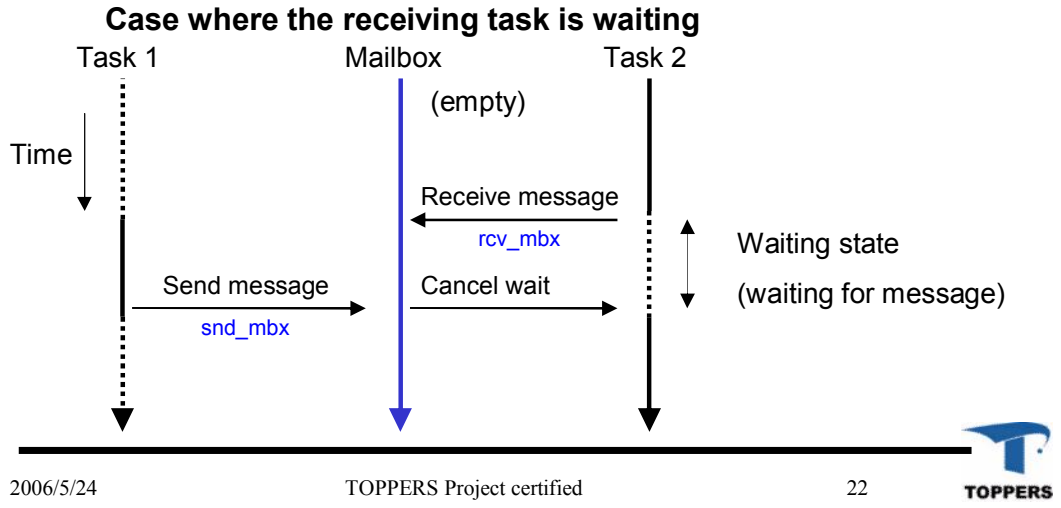
A mailbox is an object used for synchronization and communication by sending and receiving messages placed in shared memory. A mailbox has a message queue to store sent messages and a wait queue for receiving messages. The task sending a message places the message to be sent in the message queue. Task receiving the message removes the first message from the message queue. If there is no message in the message queue, the task will be in the waiting receiving state until a message is sent to the mailbox. Task waiting to receive a message will be placed in the mailbox wait queue.

Mailbox functions includes creating and deleting mailboxes, send and receive messages, and referencing state of mailboxes.

Mailbox 2

Transition of task condition

- By handing over the message, synchronization and communication is done simultaneously
  only the header address of the message is sent



Mailbox function service call specifications for μITRON4.0 is as follows:

1. Create mailbox

CRE_MBX(ID mbxid, {ATR mbxatr, PRI maxpri, VP mprihd});

ID mbxid	mailbox ID number
ATR mbxatr	mailbox attribute
	((TA_TFIFO||TA_TPRI)|(TA_MFIFO||TA_MPRI))
PRI maxpri	maximum message priority
VP mprihd	start address of area for message queue headers for each message priority

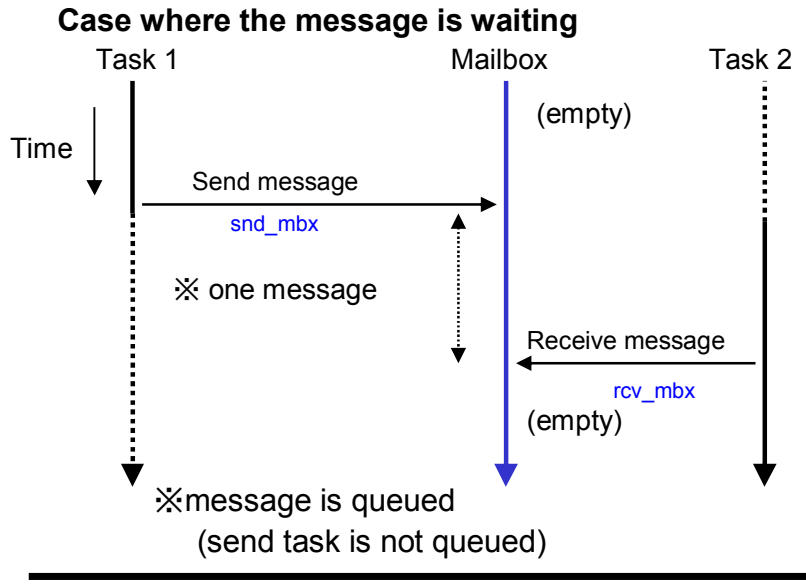
2. Send to mailbox

ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg);

ID mbxid	mailbox ID number to which message is sent
T_MSG * pk_msg	start address of message packet to be sent to mailbox

Mailbox 3

Task transition status



TOPPERS

3. Receive from mailbox

ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg);

ER ercd = prcv_mbx(ID mbxid, T_MSG **ppk_msg);

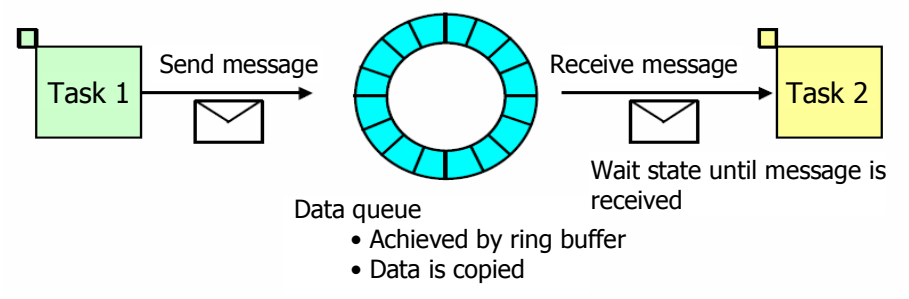
ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);

ID mbxid	mailbox ID number from which a message is received
TMO tmout	specified timeout (trcv_mbx only)
T_MSG ** ppk_msg	holding area of the message received from mailbox

Data queue 1
How does it work?

- Send and receive 1 word message (data)

  Simultaneously performs synchronization and communication



A data queue is an object used for synchronization and communication by sending and receiving one word messages. A data queue has a wait queue for sending a data element (send-wait queue), a wait queue for receiving a data element (receive-wait queue), and a data queue area to store sent data elements.

A task sending a data element places the data into the data queue. If the data queue is full, the task will be in the sending waiting state until there is room is the data queue area. Task waiting to send the data element is placed into the data queue's send-wait queue. A task receiving a data element removes one data from the data queue. If there is no data in the queue, the task will be in the receiving waiting state until the next data is sent to the data queue. Task waiting to receive data from the data queue is placed into the data queue's receive-wait queue.

Data queue functions includes creating and deleting data queues, send, force send and receive data elements to/from data queue, and referencing data queue state.

Data queue 2

Difference between mailbox and data queue

- Mailbox
  - Employs link list for data structure
  - Message length is unrestricted
  - Maximum number of messages is unrestricted (send task is not halted)
  - Needs a header
  - Only the head address of the message is sent and received, the main body is not copied
  - Data element needs to be preserved until message is received
- Data queue
  - Message length is one word
  - Maximum number of messages is fixed (send task can be placed in waiting state)
  - No need for header
  - Main body of the message is copied
  - No need to preserve data element until message is received

Data queue function service call specifications for μITRON4.0 is as follows:

1. Create data queue

CRE_DTQ(ID dtqid, {ATR dtqatr, UINT dtqcnt, VP dtq});

ID dtqid	data queue ID number
ATR dtqatr	data queue attribute (TA_TFIFO||TA_TPRI)
UINT dtqcnt	capacity of data queue area (number of data elements)
VP dtq	start address of data queue area

2. Send to data queue

ER ercd = snd_dtq(ID dtqid, VP_INT data);

ER ercd = psnd_dtq(ID dtqid, VP_INT data);

ER ercd = ipsnd_dtq(ID dtqid, VP_INT data);

ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);

ID dtqid	data queue ID number of which data element is sent
VP_INT data	data element to be sent
TMO tmout	specified timeout (tsnd_dtq only)

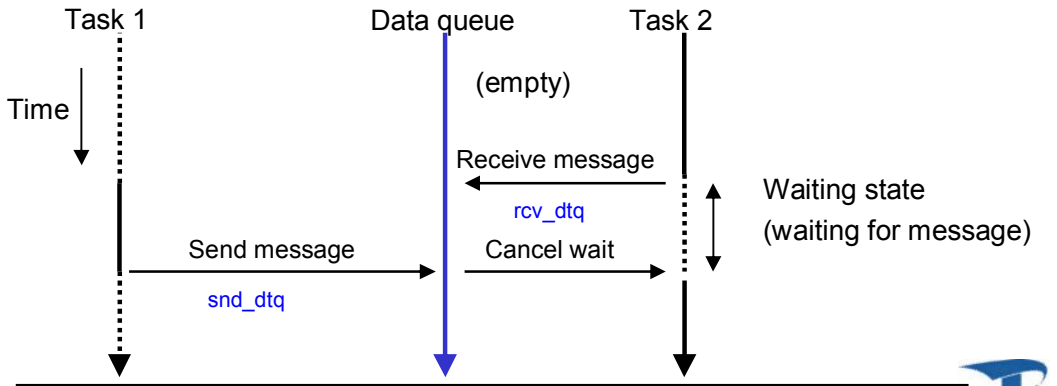
Data queue 3

Task status transition

- By handing the message, synchronization and communication is performed simultaneously

  Only an one word message is sent

Case where receive task is waiting



TOPPERS

3. Forced send to data queue

ER ercd = fsnd_dtq(ID dtqid, VP_INT data);

ER ercd = ifsnd_dtq(ID dtqid, VP_INT data);

ID dtqid	data queue ID number of which data element is sent
VP_INT data	data element to be sent to data queue

4. Receive from data queue

ER ercd = rcv_dtq(ID dtqid, VP_INT *p_data);

ER ercd = prcv_dtq(ID dtqid, VP_INT *p_data);

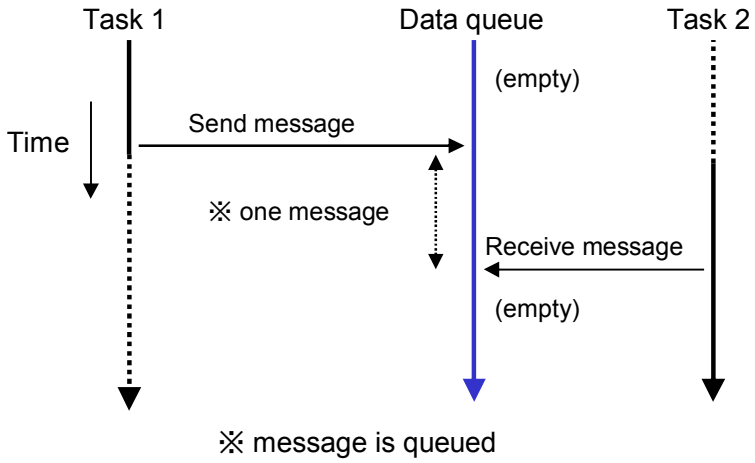
ER ercd = trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);

ID dtqid	data queue ID number from which a data element is received
TMO tmout	specified timeout (trcv_dtq)
VP_INT *p_data	holding area of data element received

Data queue 4

Task status transition

Case where a message is waiting

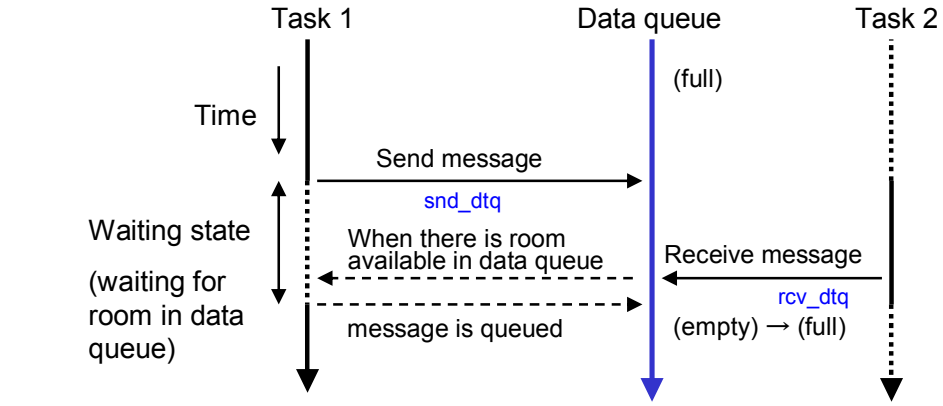


TOPPERS

Data queue 5

Task status transition

Case where the data queue is full and is waiting for room in the data queue



When data queue is full, sending task is placed in send-wait queue

TOPPERS

μITRON4.0 provided synchronization/communication functions specification

- Types of communication/synchronization functions

	Semaphore	**Event**	**Mailbox**	**Data queue**
Function	Exclusive control/ synchronization	Event notification	Synchronization/communication	Synchronization/communication
Information load	small	medium	large	large
Overhead	small	small	medium - large	large
Queued object	Resource waiting task	Event waiting task	Receive task Message	Send task Receive task Message
Service call	CRE_SEM, sig_sem, wai_sem, twai_sem, pol_sem	CRE_FLG, set_flg, clr_flg, wai_flg, pol_flg, twai_flg	CRE_MBX, snd_mbx, rcv_mbx, prcv_mbx, trcv_mbx	CRE_DTQ, snd_dtq, psnd_dtq, tsnd_dtq, fsnd_dtq, rcv_dtq, prcv_dtq, trcv_dtq

When constructing a system, if multiple communication and synchronization functions within the system is used the system will become complex. Therefore unless there is reason to do otherwise, it is recommended to select communication and synchronization function so that the system construction is simple.

Reconstruction using eventflags

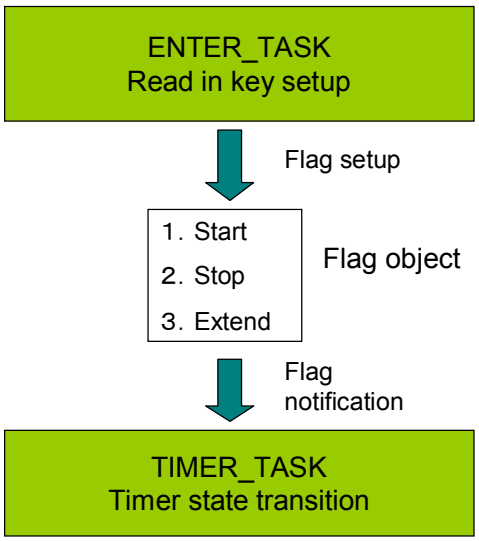
1. Implementing the instant noodle timer
2. Review
3. Synchronization and communication
4. **Reconstruction using eventflags**
5. Summary

Let us reconstruct the RTOS instant noodle timer so that task communication is done using eventflags. By using eventflags, module cohesion will be weakened.

Task communication utilizing eventflags 1

Reconstructing the section using shared memory

- Using flag object, construct a command notification structure from ENTER_TASK TO TIMER_TASK
- Command is communicated by the eventflag service call



TOPPERS

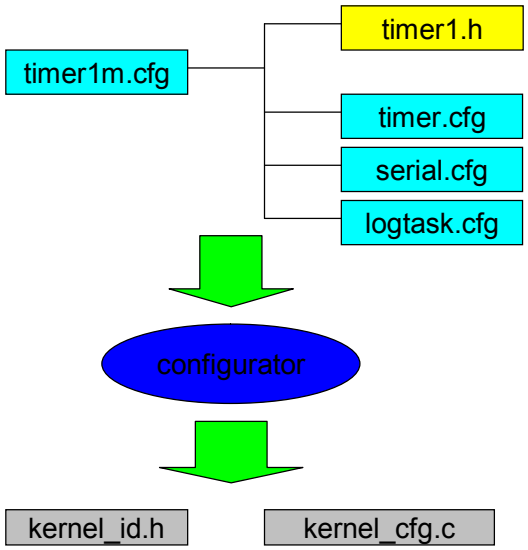
How does it differ from when using shared memory?

1. With shared memory, global data is referenced and set by two tasks. Therefore when a change occurs to the module, area implementation must be considered by the two tasks. Compared to this, implementation using eventflags manages the data using specific objects, making interface construction clear. In this example, it may look as if there is no large difference, but in cases where the interface is more complex, benefits become clear.
2. By using eventflags, because it is supported as a synchronization/communication function, starting the received task right after an event occurrence is possible. Compared to this, when only shared memory is used, receiving tasks must poll for event occurrences and therefore overhead eventuates.

Task communication utilizing eventflags 2

Configuration setup

- Task resource with the TOPPERS/JSP kernel is defined by the configuration file
- Resource is converted to two sources, kernel_id.h and kernel_cfg.c, by the configurator
- CRE_FLG is defined to timer1m.cfg



By using an editor, change the configuration file. By executing *make depend* from the setting within *makefile*, the configurator (../../cfg/cfg.exe) will start up and create kernel_id.h and kernel_cfg.c from timer1m.cfg.

Reference notes from eventflag#2 for CRE_FLG.

For the eventflag ID, set label of your preference. The eventflag attribute is (TA_TFIFO|TA_WSGL). Initial value of the eventflag bit pattern is 0. TA_TFIFO is the setting for FIFO order and TA_WSGL means that only a single task is allowed to be in the waiting state for an eventflag.

Task communication utilizing eventflag 3

Send task implementation

- When sending an event, send by using the set_flg service call
- Three events can be used as is

```
event |= EVT_TIMER_START;
```

```
set_flg(eventflag ID, EVT_TIMER_START);
```

EVT_TIMER_START	Start timer
EVT_TIMER_STOP	Stop timer
EVT_TIMER_COUNT	Extend timer time

First it is necessary to define the bit patterns for communication.

Here we will use the same definition that was used for defining shared memory event.

Also the eventflag ID specified in the configuration file will be automatically converted by the configurator as set within kernel_id.h, therefore that label can be used as the eventflag ID.

When sending the event, use the set_flg service flag to send the event.

```
ER set_flg (ID flgid, FLGPTN setprn);
```

ID	flgid	ID number of the eventflag to be set
FLGPTN	setptn	bit pattern to set

In this case, there will be no error occurrence so there is no need for an error check.

Task communication utilizing eventflag 4

Receive task implementation

- Wait for eventflag with timeout or wait for eventflag by polling is utilized
- When the service call outcome is E_OK, bit pattern of flgptn is turned ON
- Flag can be cleared by clr_flg after inheriting the flag

```
#define EV_TIME        (start|stop|extend)
/* description within timer task*/
FLGPTN flgptn;         /* flag pattern */
TMO tmout;             /* timeout(ms)*/
ER    ercd;            /* service call result */
                :
tmout = timeout;
ercd =
twai_flg(EVT_FLGID,EV_TIME,TWF_ORW,&flgptn,tm
out);
If(ercd == E_OK){
  clr_flg(EVT_FLGID, ~flgptn);
                :
}
```


EV_TIME is the OR value of the three wait bits.

twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *flgptn, TMO tmout);

ID	flgid	ID number of the eventflag to wait for
FLGPTN	waiptn	wait bit pattern
MODE	wfmode	wait mode
FLGPTN	flgptn	receive pattern holding area
TMO	tmout	specified timeout (ms)

When wait mode has TWF_ANDW set, the release condition requires all the bits in the wait pattern to be set. With TWF_ORW, the release condition only requires at least one bit in the wait pattern to be set.

Acquiring a flag using the eventflag wait service call does not clear the flag that has been set. Therefore the clr_flg service call is needed for clearing. With timeout settings, the tmout value changes depending on the wait condition value set.

TMO_POL(=0)	same as pol_flg; specifies polling setting
TMO_FEVR(=-1)	same as wai_flg; endless timeout duration

tmout value is milliseconds.

Summary

1. Implementing the instant noodle timer
2. Review
3. Synchronization and communication
4. Reconstruction using eventflags
5. **Summary**

Let us summarize. Purchase the board and try working the exercises.

Task communication/synchronization design 1
Communication via shared memory

- Place data to be shared between tasks in shared memory
- Exclusive control is needed (excluded if the shared data can be read/written at once)
  - disable interrupt/dispatch
  - utilize semaphore/mutex
  - change task priority
- A separate mechanism may be necessary to notify the occurrence of an event
  - task start/wakeup
  - utilize eventflag/condition variable

Here is some explanation on how to construct task communication functions of system designs. Communication methods are largely divided into “communication by shared memory” and “communication by message”.

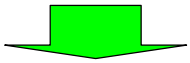
With communication by shared memory, data to be sent/received between tasks is secured in global memory areas and/or shared memory. By read/write of data placed in these areas, communication is made between tasks. Since each task and interruptions are asynchronous actions, if a task reads in data before the other task has written in the data, erroneous information may be received. Therefore by using, for example semaphores for external controls, it is possible to prevent these situations.

Also, if there is a change in the event occurrence or a need arise where action is immediately necessary, it is necessary to run the receiving task using task start/wake-up and/or eventflags.

Task communication/synchronization design 2

Communicating by message

- Realization of task communication through message
- Structure is needed for message communication
- Types of message communication structures:
  - ・synchronous message communication or asynchronous message communication
  - ・handover pointer or copy pointer
  - ・sequence of queuing (FIFO, priority)
  - ・action when there is no message and when message is full
  - ・packet unit exists or does not exist



Good understanding of the RTOS message communication structure (may be multiple structures) is necessary

TOPPERS

For “communication by message”, construction is done by utilizing message functions such as mailboxes and data queues. The message content needs to be examined carefully when setting the system. Message structure can be self made if external control is managed. Consider the whole system and construct in the best possible method.

Task communication/synchronization design 3

Shared memory vs. Message communication

- Basically, if it can be accomplished using one method, then it is possible to be done in the other as well (task wait condition must be taken into consideration)
  - In general, the shared memory method has a smaller overhead
  - Message method is easier to handle for system verification. [e.g. easier to separate problems]
- Depending on the circumstance, one method may be advantageous/convenient compared to the other
  - Shared memory is advantageous when the flow of information is 1:n and timing of when the information is needed is ambiguous
  - Message method is convenient when queuing of information is necessary

System designing becomes complex when both methods are utilized, and careful deliberation is prudent

APPENDIX

Characteristics of TOPPERS/JSP kernel

μITRON4.0 conformed standard profile

- Source code is easy to read and reconstruct
- Easy porting to targets structure
- High efficiency performance and low RAM usage
- Linux / Windows simulation tool
- Possible to construct using only free software, including development tool

The TOPPERS/JSP kernel is a real-time kernel that is in accordance with μITRON4.0 specification. It is also the first development result by the TOPPERS project. JSP is an acronym for Just Standard Profile, and as the name shows, is implemented in accordance with the μITRON4.0 specification standard profile rule. The latest JSP release can be downloaded from http://www.toppers.jp/index.html.

Main features are as follows:

•Easy to read and reconstructible source code

Keeping in mind that utilization would be for research and development, emphasis was put on easy to read and reconstructible source codes. The algorithm, although easy to read, is not inefficient. Rather for example, by using heap structure to manage time events, complex but efficient algorithms are employed.

•Porting to other targets is easy to do

C language is used for most parts of the kernel. Clear separation of target independent and dependent parts are made, therefore making porting to other target processors and systems is made easy. There are reports where, if the target processor architecture is familiar, that porting was accomplished in three days.

•High efficiency performance and low RAM usage

For a kernel where most parts are written in C, high efficiency performance and low RAM usage is realized.

•Simulation environment on Linux and Windows

Simulation environments for JSP kernel to run on Linux and Windows are available. These simulation environments switches multiple tasks within one process in Linux/Windows. It is the best suited for proto-type development of embedded systems, logic level verifications, and real-time learning experiences.

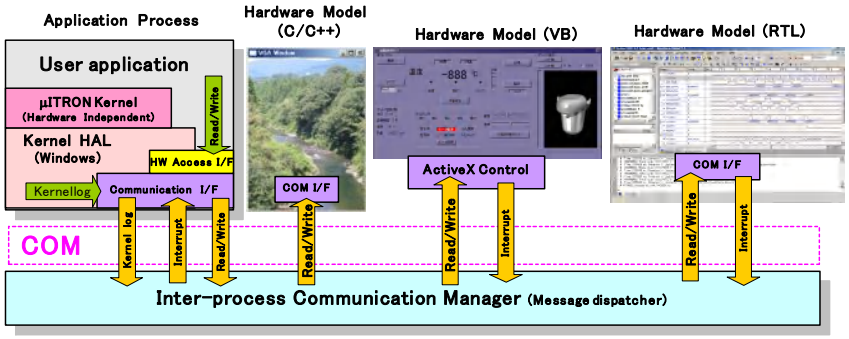
•It is possible to construct systems solely using free software

GCC and GNU development tool are standard software development tools. Therefore the user can acquire not only the kernel itself, but also the development tool free and develop a system.

Simulation without hardware

Development employing Windows simulator

- Enable to implement, simulate, and test on Windows
- Same source code can be used as if executing on real hardware
- Source code in C++ is made available



TOPPERS

Windows simulator supplements the TOPPERS/JSP kernel Release1.4 Windows version kernel, and provides the interface to the device and system log.

These parts which compose the interface are called device drivers, and exist on Windows through the COM interface. With the interface between device driver and Visual BASIC, the device control program controls the interface with the device and the log watcher control program maintains the system log. Installation directions for these programs are noted in the TOPPERS/JSP kernel Release1.4 doc/windows.txt.

Standard JSP kernel development tool 1
LINUX and Cygwin

- CPU corresponding to TOPPERS/JSP kernel and development board dependent source is located in the config directory
- The above mentioned are stored accordingly to the following naming rule:
  - GNU development tool

    tool name is abbreviated

    h8/akih8_3069f [GNU environment]
  - Other development tool besides GNU

    tool name is noted after the CPU

    CPU name - tool name / board name

    m16c-renesas/oaks16_mini [Renesas Technology Corp environment]

GNU development tool can be developed on Linux and/or Cygwin

The basic TOPPERS/JSP kernel development tool is one where GNU is utilized.

There is a naming rule to the CPU and board correspondences under the config directory. If the tools name is omitted, it specifies that they should be constructed under the GNU environment.

In order to use the GNU compiler and linker, an UNIX executable environment such as LINUX is necessary. With Windows, an UNIX executable environment such as Cygwin is necessary.

In these cases, Perl and Gnu-make is used for construction. Therefore it will be wise for the development manager to hold knowledge of these languages.

Standard JSP kernel development tool 2

Developing on Cygwin

- Install Cygwin onto Windows
- Install the target GNU development tool onto Cygwin

Install BINUTILS, GCC, GDB, NEWLIB

- Build the configuration tool(s)
- Using configuration script, create *Makefile* for target and sample programs
- After defining dependency relations, build the target

Development tool defined

Constructing the development tool

- Development tools can be constructed on your computer as well

  In the CD-ROM attached to the board, a GNU development tool (Cygwin, BINUTILS, GCC, NEWLIB) is contained. Follow the installation instructions to install onto your computer

- There is a sample source on the CD-R

  Sample source for this seminar is contained on the CD-R.

  Construction of environment needed for this seminar can be made from this.