Message queues

Introduction

- **Semaphore는 synchronization, mutual exclusion 지원.**
- **But, inter-task message exchange 불가.**

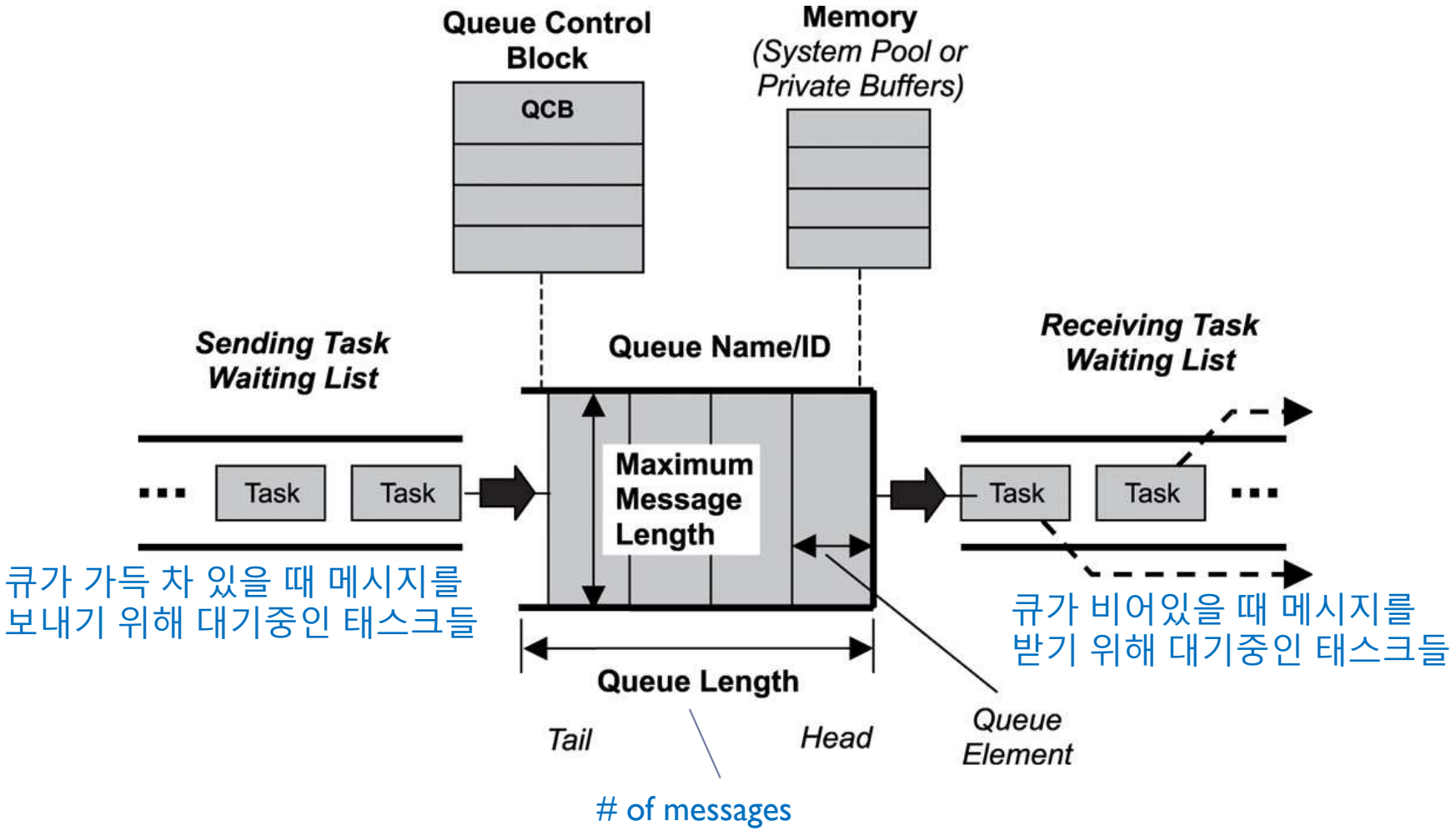
- **For inter-task data communication, RTOS provides**
  - A message queue object
  - Message queue management services

Defining message queues

- **Message queue**
  - A buffer-like object which tasks and ISRs send/receive messages to communicate and synchronize with data.
  - Temporarily holds messages from sender until receiver read them.
  - This temporary buffering decouples a sending and receiving task.
    - 동시에 메시지 송수신을 해야 할 필요가 없다.

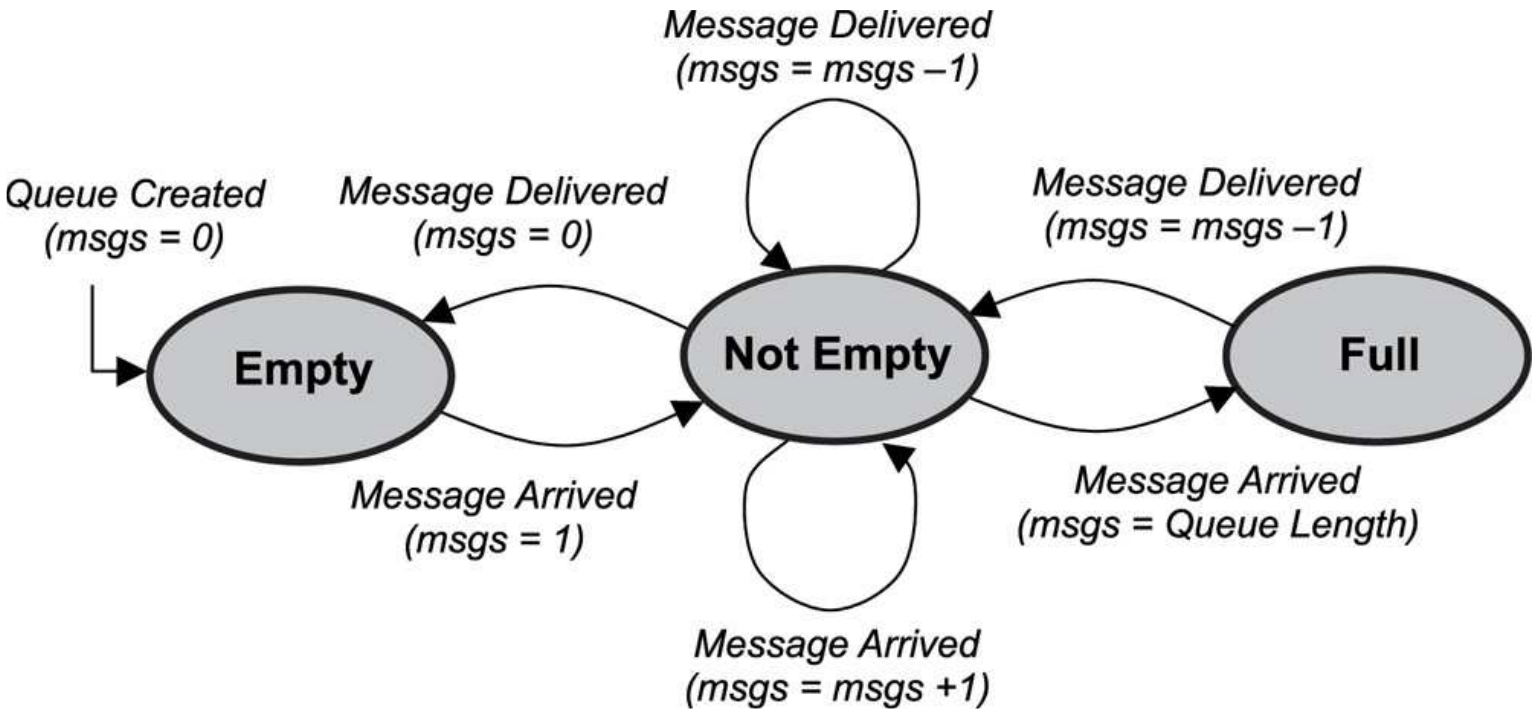
Defining message queues

- **Data structures for message queues**



Message queue states

- **Static diagram of message queue**

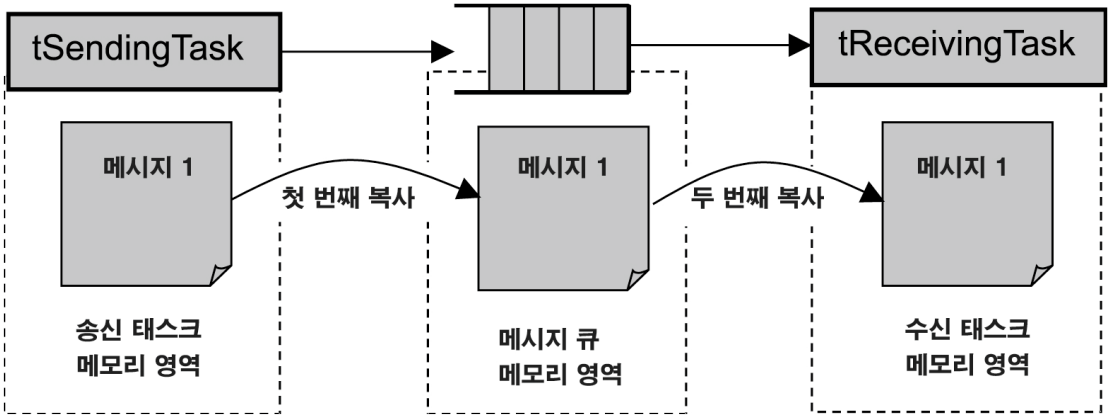


Message queue content

- **Example of messages**
  - Temperature value from a sensor
  - A bitmap to draw on a display
  - A text message to print to an LCD
  - A keyboard event
  - A data packet to send over the network

Message queue content

- Different sizes of message length
- Copy overhead of a message



- ➔ use “pointer” instead of data itself.

Message queue storage

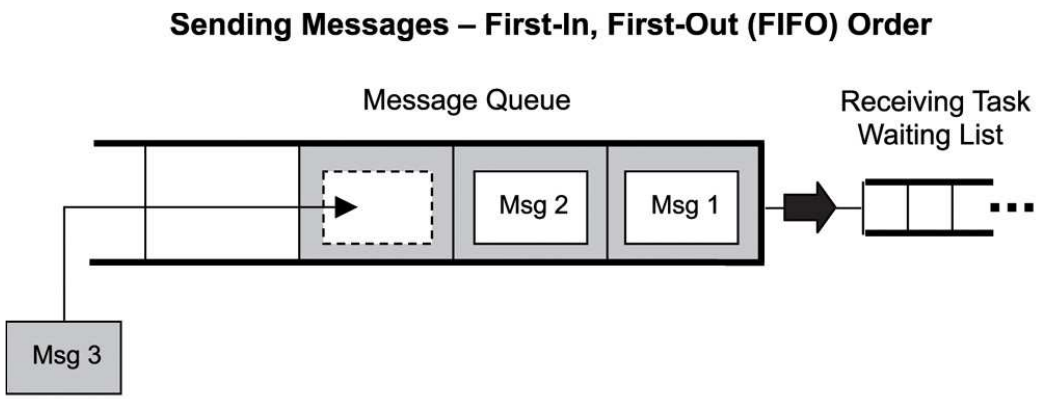
- **How to store message in memory**
  - System pools
    - Messages of all queues are stored in one shared memory.
    - (+) Save on memory use
    - (-) a message queue with large messages may use most of the pooled memory.
  - Private buffers
    - Use separate memory areas for each message queue
    - (+) room is available for all message queues ➔ fair
    - (-) requires a lot of memory.

Message queue operations

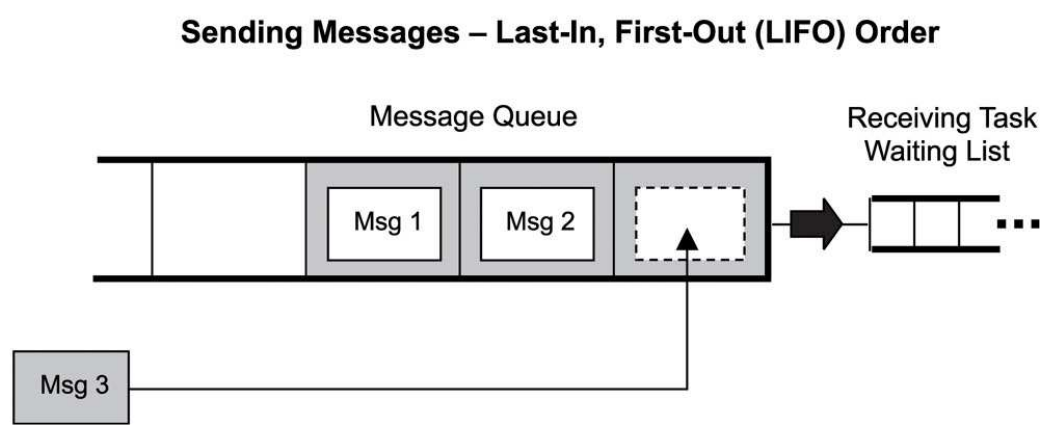
- **Typical message queue operations**
  - Creating and deleting message queues
  - Sending and receiving messages
  - Obtaining message queue information

Sending messages

- **Sending messages – FIFO or LIFO order**



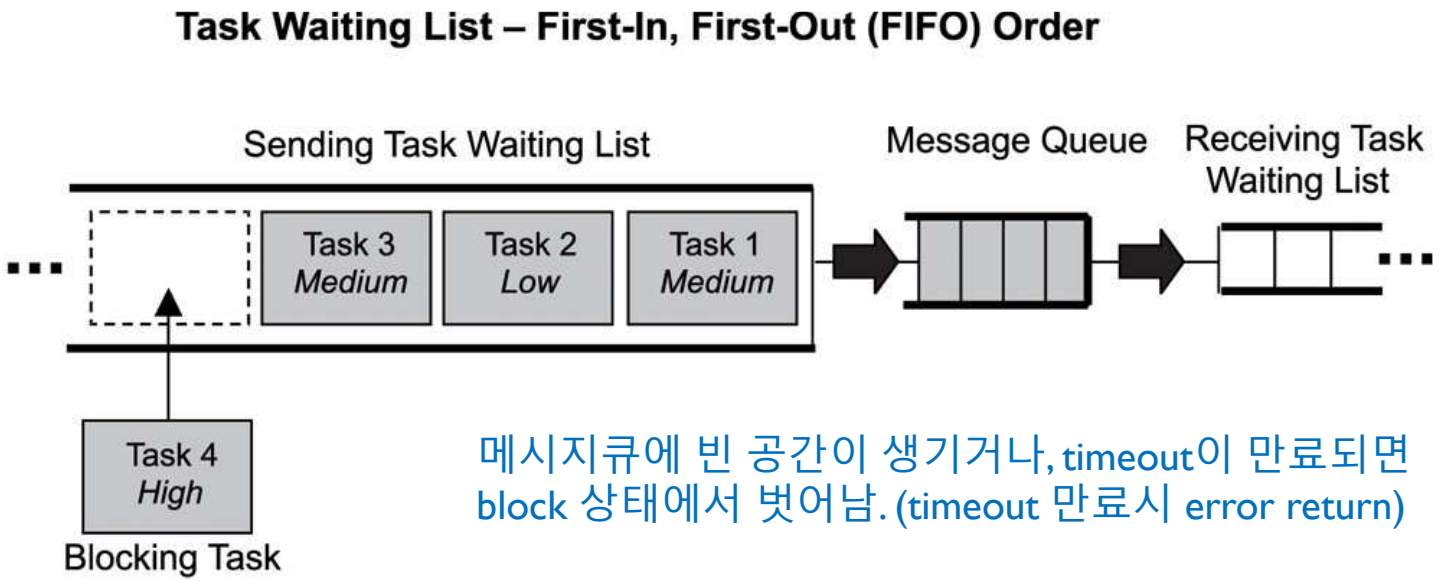
A message is typically filled from head to tail in FIFO order.



Urgent messages are gone to straight to the head of the queue.
eg. ISRs

Sending messages

- **Messages are sent to a message queue.**
  - Not block (ISRs and tasks)
    - If a message queue is full, sender returns with an error and task or ISR continues executing.
    - ISR cannot block.
  - Block with a timeout (tasks only)
  - Block forever (tasks only)



메시지큐에 빈 공간이 생기거나, timeout이 만료되면 block 상태에서 벗어남. (timeout 만료시 error return)

Receiving messages

- **Tasks receive message with different blocking policies**
  - Not blocking
  - Blocking with a timeout
  - Blocking forever

- **Blocking occurs due to the empty message queue.**
  - When message queue is full
    - ➔ the sending task-waiting list start to fill.
  - When message queue is empty
    - ➔ the receiving task-waiting list start to fill.
  - Receiving and sending rates are different.

Receiving messages

- **Messages can be read from the head of message queue.**
  - Destructive read
    - When receives a message, the message is permanently removed.
  - Non-destructive read
    - Task peeks at the message without removing it.

Typical message queue use

- **Typical usage of message queue**
  - Non-interlocked, one-way data communication
  - Interlocked, one-way data communication
  - Interlocked, two-way data communication
  - Broadcast communication
  - …

Non-interlocked, one-way data communication

KWANGWOON UNIVERSITY

- **The simplest message-based communication**
  - tSourceTask simply sends a message.
    - It does not require acknowledgement from tSinkTank.
  - If tSinkTask has a higher priority
    - tSinkTask runs first ➔ tSinkTask blocks ➔ tSourceTask sends a message ➔ tSinkTask starts to execute again.
  - If tSinkTask has a lower priority
    - tSourceTask runs first ➔ tSourceTask fills the message queue ➔ tSourceTask blocks ➔ tSinkTask wake up and take message

Non-interlocked, one-way data communication

KWANGWOON UNIVERSITY

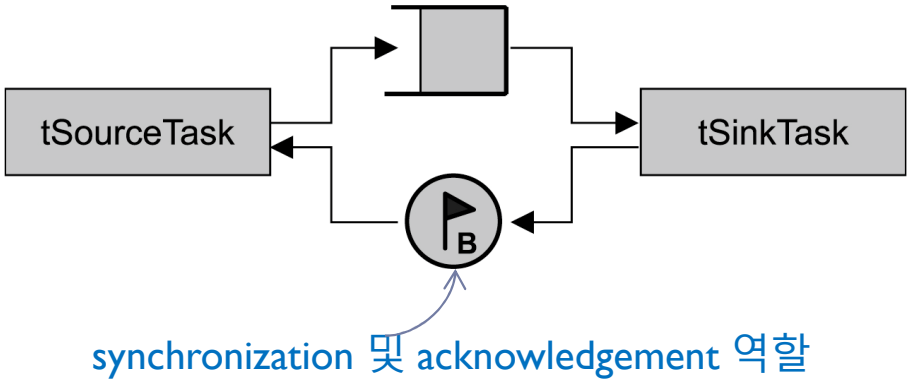
- **Pseudo code**

```
tSourceTask ( )
{
                :
                send message to message queue
                :
}

tSinkTask ( )
{
                :
                receive message from message queue
                :
}
```

Interlocked, one-way data communication

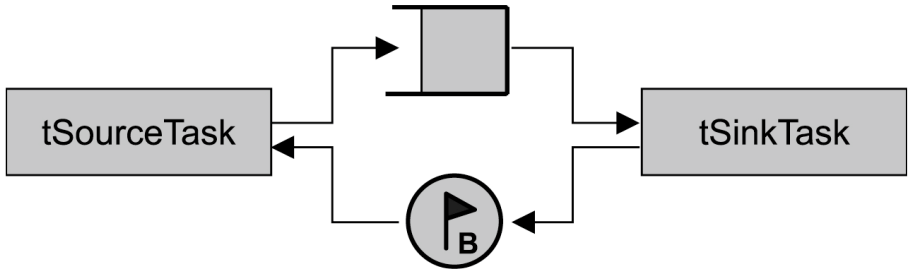
- **Interlocked communication**
  - Sending task sends a message and waits an acknowledgement.
  - Useful for reliable communications or task synchronization.
  - tSourceTask and tSinkTask use
    - A binary semaphore initially set to 0
    - A message queue with a length of 1
  - tSourceTask sends message and blocks on the binary semaphore ➔ tSinkTask receives the message and increments the binary semaphore ➔ tSourceTask wakes up ➔ … (continuous loop)



Interlocked, one-way data communication

- **Pseudo code**

```
tSourceTask ( )
{
                :
                send message to message queue
                acquire binary semaphore
                :
}
tSinkTask ( )
{
                :
                receive message from message queue
                release binary semaphore
                :
}
```

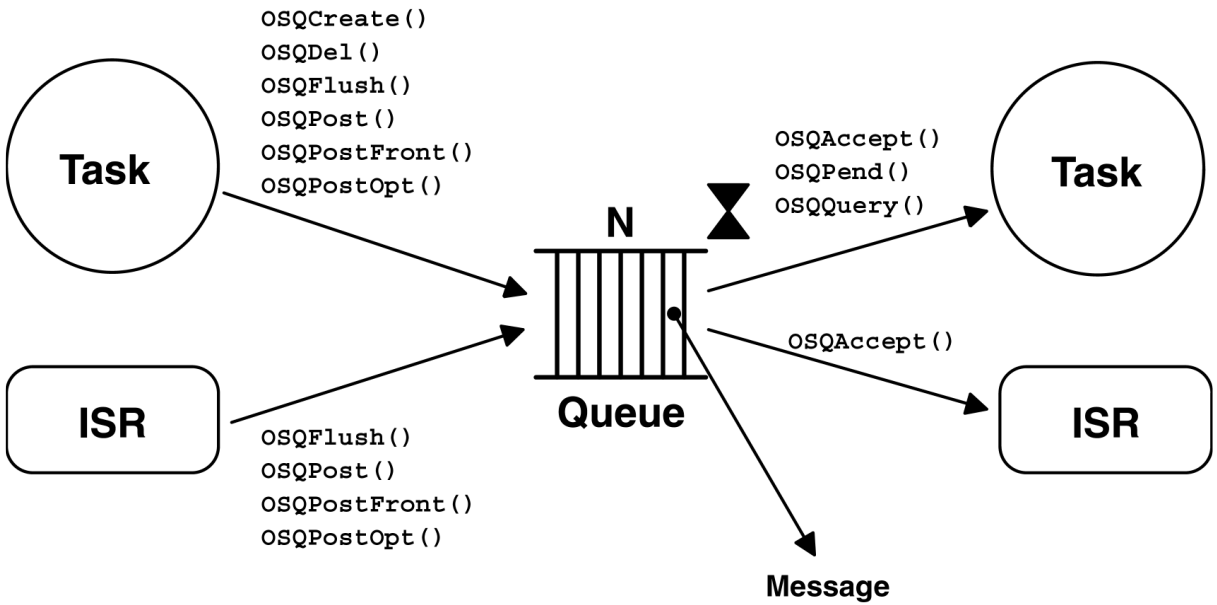


Case study: uCOS-II

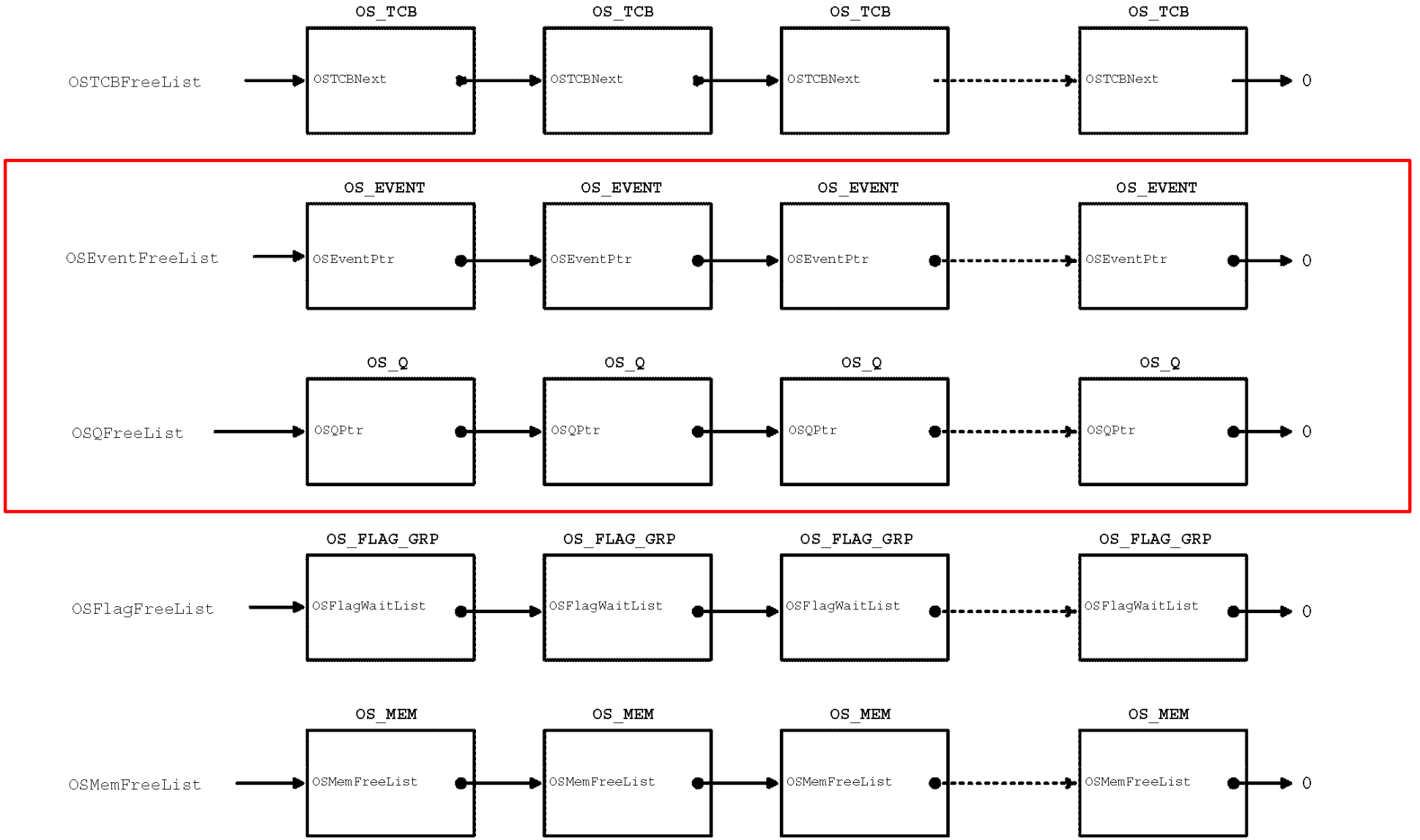
Message queues

메시지큐

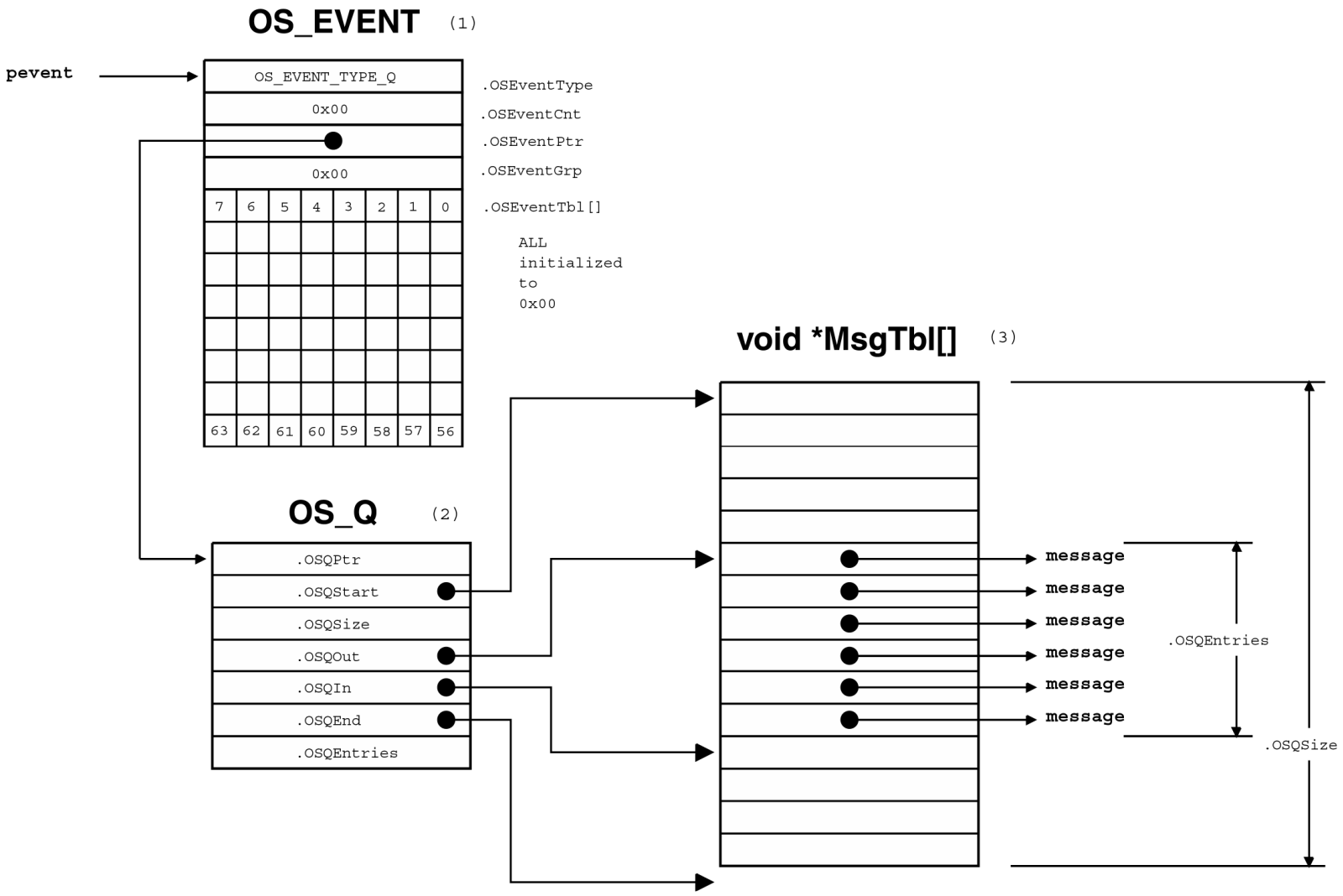
- 여러 개의 메시지를 전달할 수 있는 IPC
  - FIFO, LIFO 모두 가능
- 메시지큐의 크기는 자유
  - 응용 프로그램이 메시지 주소를 저장할 배열 공간을 확보한 후에, 메시지큐 생성.



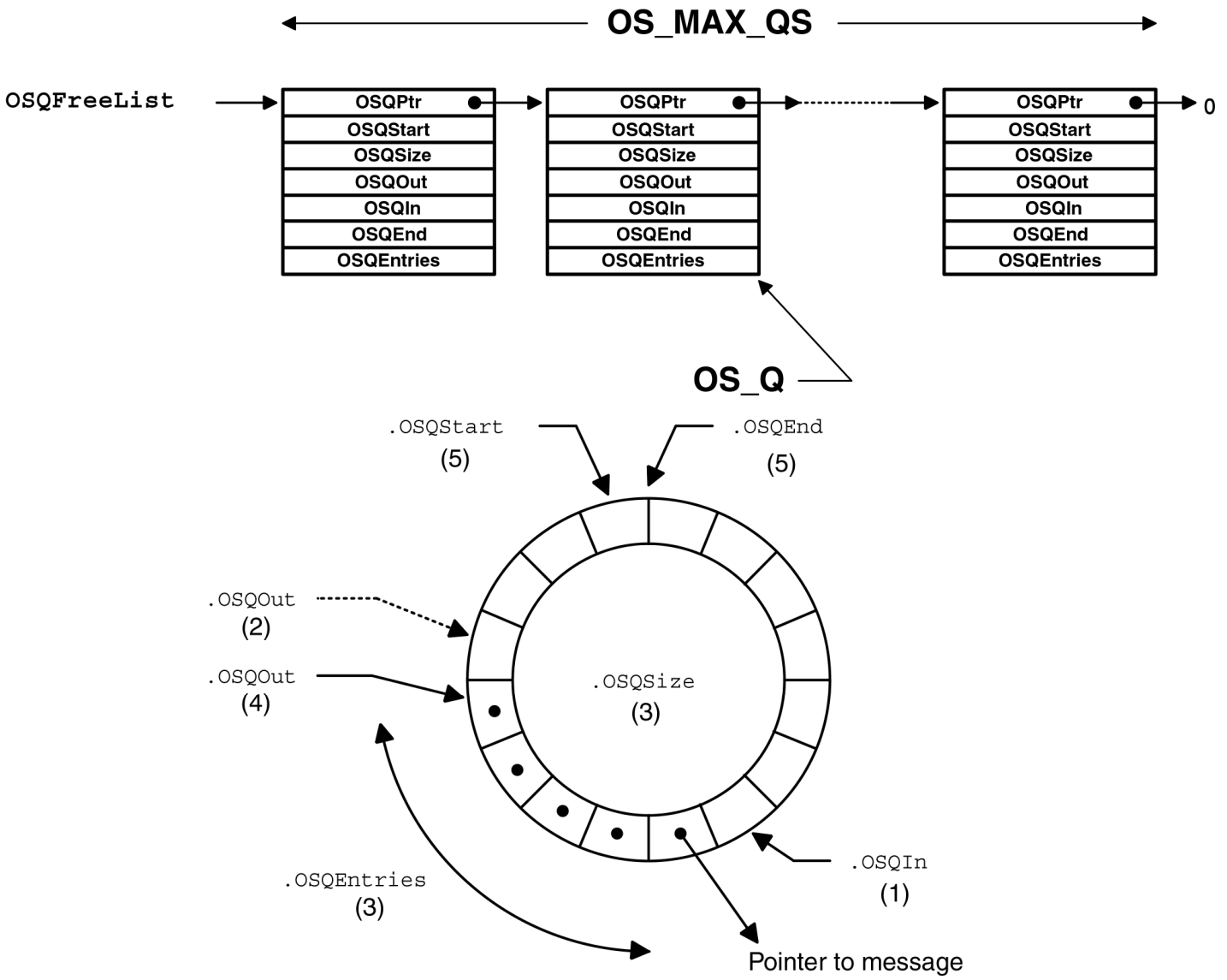
OSInit() review



메시지큐


OS_EVENT (1)
pevent
OS_EVENT_TYPE_Q
0x00
0x00
.OSEventType
.OSEventCnt
.OSEventPtr
.OSEventGrp
.OSEventTbl[]
7 6 5 4 3 2 1 0
63 62 61 60 59 58 57 56
ALL initialized to 0x00
OS_Q (2)
.OSQPtr
.OSQStart
.OSQSize
.OSQOut
.OSQIn
.OSQEnd
.OSQEntries
void *MsgTbl[] (3)
message
message
message
message
message
message
.OSQEntries
.OSQSize

OS_Q 관리


OS_MAX_QS
OSQFreeList
OSQPtr
OSQStart
OSQSize
OSQOut
OSQIn
OSQEnd
OSQEntries
0
OS_Q
.OSQStart
(5)
.OSQEnd
(5)
.OSQOut
(2)
.OSQOut
(4)
.OSQSize
(3)
.OSQIn
(1)
.OSQEntries
(3)
Pointer to message

메시지큐 구조체

- **uC/OS-II에서 메시지큐는 EventControlBlock 및 OS_Q 구조체를 이용하여 구현됨**

```
typedef struct {
    INT8U   OSEventType;           /* Event type                          */
    INT8U   OSEventGrp;            /* Group for wait list                 */
    INT16U  OSEventCnt;            /* Count (when event is a semaphore)*/
    void   *OSEventPtr;            /* Ptr to message or queue structure*/
    INT8U   OSEventTbl[OS_EVENT_TBL_SIZE]; /* Wait list for event         */
} OS_EVENT;
```

  - OSEventType
    - 메시지큐의 경우, OS_EVENT_TYPE_Q 로 지정
  - OSEventPtr
    - 메시지큐의 경우, 메시지 큐 (OS_Q) 주소 저장
  - OSEventTbl[], OSEventGrp
    - 준비 리스트와 동일한 방법으로 메시지큐 대기 리스트 표현
  - OSEventCnt
    - 메시지큐의 경우, 사용하지 않음

메시지큐 구조체

```
typedef struct os_q {
    struct os_q   *OSQPtr;
    void          **OSQStart;
    void          **OSQEnd;
    void          **OSQIn;
    void          **OSQOut;
    INT16U          OSQSize;
    INT16U          OSQEntries;
} OS_Q;
```

- OSQPtr- 다음 OS_Q에 대한 포인터
- OSQStart - 메시지 주소를 저장할 포인터 배열의 시작 주소
- OSQEnd - 메시지 주소를 저장할 포인터 배열의 끝 주소
- OSQIn - 포인터 배열에서 메시지를 FIFO 방식으로 추가할 위치
- OSQOut - 포인터 배열에서 메시지를 꺼낼 위치
- OSQSize - 포인터 배열의 크기 (메시지 최대 저장 개수)
- OSQEntries - 현재 저장된 메시지 개수

API

- **API**
  - OSQCreate() // 메시지큐 생성 및 초기화
  - OSQDel() // 메시지큐 삭제
  - OSQPend() // 메시지 수신
  - OSQPost() // 메시지 송신 (FIFO)
  - OSQPostFront() // 메시지 송신 (LIFO)
  - OSQPostOpt() // 메시지 브로드캐스트
  - OSQAccept() // 메시지 수신 시도
  - OSQFlush() // 메시지큐 비움 (메시지 버림)
  - OSQQuery() // 메시지큐 정보 획득

Creating a Q, OSQCreate()

- **OS_Q.C / Kernel**

- **OS_EVENT *OSQCreate (void **start, INT16U size)**

  - start : 메시지 주소들을 저장할 포인터 배열의 시작 주소
  - size : 포인터 배열의 크기

  - 메시지큐 구조체를 생성하고, 큐 구조체가 포인터 배열을 가르키도록 초기화
  - 생성된 메시지큐 반환
  - ISR은 호출할 수 없음

Send a Msg (FIFO), OSQPost()

- **OS_Q.C / Kernel**

- **INT8U OSQPost (OS_EVENT *pevent, void *msg)**

  - pevent : 대상 메시지큐
  - msg : 메시지

  - 메시지를 송신한다 (FIFO).
  - 대기 중인 태스크가 있으면, 최상위 우선순위 태스크에게 메시지 전송
  - 대기 중인 태스크가 없으면, 메시지큐에 메시지 저장 (FIFO). 만약 메시지큐가 full 이면 에러 반환

Send a Msg (LIFO), OSQPostFront()

- **OS_Q.C / Kernel**

- **INT8U OSQPostFront(OS_EVENT *pevent, void *msg)**

  - pevent : 대상 메시지큐
  - msg : 메시지

  - 메시지를 송신한다 (LIFO).
  - 대기 중인 태스크가 있으면, 최상위 우선순위 태스크에게 메시지 전송
  - 대기 중인 태스크가 없으면, 메시지큐에 메시지 저장 (LIFO). 만약 메시지큐가 full 이면 에러 반환

Broadcast a Msg, OSQPostOpt()

- **OS_Q.C / Kernel**

- **INT8U OSQPostOpt (OS_EVENT *pevent, void *msg, INT8U opt)**
  - pevent : 대상 메시지큐
  - msg : 메시지
  - opt : broadcast 옵션

  - 메시지를 송신한다.
  - 브로드캐스트 옵션이 설정되어 있으면, 대기 중인 모든 태스크에게 메시지 전송
  - 브로드캐스트 옵션이 설정되어 있지 않으면, 옵션(OS_POST_OPT_FRONT)에 따라, OSQPost() 또는 OSQPostFront()로 동작
  - 대기 중인 태스크가 없으면, 메시지큐에 메시지 저장. 만약 큐가 full이면 에러 반환

Receive a Msg, OSQPend()

- **OS_Q.C / Kernel**

- **void *OSQPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)**
  - pevent : 대상 메시지큐
  - timeout : 메시지 수신 실패 시, timeout tick 수
  - err : 메시지큐 연산의 결과값

  - 수신한 메시지를 반환한다 (수신 실패 시에는 0 반환).
  - 수신 실패 시, timeout ticks 만큼, 메시지큐 대기 리스트에서 블록 상태로 대기
  - timeout이 0으로 호출되면, 수신 실패 시, 수신 성공할 때까지 무한정 대기
  - ISR은 호출할 수 없음

Try to Receive a Msg, OSQAccept()

- **OS_Q.C / Kernel**

- **void *OSQAccept (OS_EVENT *pevent)**

  - pevent : 대상 메시지큐

  - 메시지 수신을 시도한다.
  - 수신 실패하는 경우, 대기하지 않고, 바로 return.
    - return 값이 존재이면, 태스크가 메시지를 수신했음을 의미
    - return 값이 0이면, 태스크가 메시지를 수신 실패했음을 의미

Flush Q, OSQFlush()

- **OS_Q.C / Kernel**
- **INT8U OSQFlush (OS_EVENT *pevent)**
  - pevent : 대상 메시지큐
  - 메시지큐의 모든 메시지를 버림
    - 내부적인 동작은 메시지큐를 처음 생성할 때의 상태와 같이 초기화

Get the status of a Q, OSQQuery()

- **OS_Q.C / Kernel**

- **INT8U OSQQuery (OS_EVENT *pevent, OS_Q_DATA *pdata)**

  - pevent : 대상 메시지큐
  - pdata : 메시지큐 상태 정보를 복사할 메모리 공간

  - 메시지큐의 정보를 pdata로 복사
    - 대기 리스트
    - 메시지큐 크기
    - 메시지 개수
    - 꺼낼 메시지 주소

Delete a Q, OSQDel()

- **OS_Q.C / Kernel**

- **OS_EVENT *OSQDel (OS_EVENT *pevent, INT8U opt, INT8U *err)**
  - pevent : 대상 메시지큐
  - opt : 옵션
  - err : 수행 결과

  - 메시지큐를 삭제
  - option이 OS_DEL_NO_PEND이면, 대기 태스크가 없는 경우에만 메시지큐 삭제
  - option이 OS_DEL_ALWAYS이면, 대기 태스크가 있는 경우에도 메시지큐 삭제.
    - 대기 태스크들은 모두 wakeup 됨

Example codes

Message queues

문제 1

- **다음의 태스크를 생성하여, 각 task가 schedule된 내용을 “log.txt” 에 저장하라.**
  - 메시지큐의 크기는 100으로 가정
  - Logging Task
    - log.txt 생성
    - 다른 태스크가 “메시지큐”를 통해 전달하는 메시지를 log.txt에 기록
    - 우선순위 0 (가장 높은 우선순위)
    - 메시지를 전달
  - 일반 Task (우선순위 10, 20)
    - current clock tick, 자신의 priority, “schedule” 의 내용을 logging task에게 “메시지큐”로 전송
    - 임의의 클록 수 (1-5) 만큼 sleep
    - 1-2 과정 반복

코드

```
#include "includes.h"
#include <time.h>

#define  TASK_STK_SIZE                  512
#define  N_TASKS                        2
#define  N_MSG                          100

OS_STK    TaskStk[N_TASKS][TASK_STK_SIZE];
OS_STK    LogTaskStk[TASK_STK_SIZE];

OS_EVENT *msg_q;                                    // (1)
void          *msg_array[N_MSG];                    // (2)


void  LogTask(void *data);
void  Task(void *data);
void  CreateTasks(void);
```

코드

```
int  main (void)
{
      OSInit();
    CreateTasks();                                                    // (3)
    msg_q = OSQCreate(msg_array, (INT16U) N_MSG);                     // (4)

    if (msg_q == 0)
    {
            printf("creating msg_q is failed₩n");
            return -1;
    }
     OSStart();
     return 0;
}

void  CreateTasks (void)                                              // (5)
{
    OSTaskCreate(LogTask, (void *) 0, &LogTaskStk[TASK_STK_SIZE - 1], (INT8U) (0));
    OSTaskCreate(Task, (void *) 0, &TaskStk[0][TASK_STK_SIZE - 1], (INT8U) (10));
    OSTaskCreate(Task, (void *) 0, &TaskStk[1][TASK_STK_SIZE - 1], (INT8U) (20));
}
```

코드

```
void  LogTask (void *pdata)                                    // (6)
{
    FILE     *log;
    void     *msg;
    INT8U  err;

    log = fopen("log.txt", "w");                               // (7)

    for (;;) {
            msg = OSQPend(msg_q, 0, &err);                     // (8)
            if (msg != 0)
            {
                    fprintf(log, "%s", msg);                   // (9)
                    fflush(log);
            }
    }
}
```

코드

```
void  Task (void *pdata)                                        // (10)
{
    INT8U  sleep, err;
    char   msg[100];

    srand(time((unsigned int *) 0) + (OSTCBCur->OSTCBPrio));    // (11)

    for (;;) {
            sprintf(msg, "%4u: Task %u schedule\n", OSTimeGet(),
                                OSTCBCur->OSTCBPrio);           // (12)
            err = OSQPost(msg_q, msg);                          // (13)
            while (err != OS_NO_ERR)
            {
                    err = OSQPost(msg_q, msg);
            }
            sleep = (rand() % 5) + 1;
            OSTimeDly(sleep);                                   // (14)
    }
}
```

문제 2

- **문제1의 내용을 다음과 같이 수정하고 수행 결과를 관찰한다.**
  - Log Task
    - for 문 안에서, 메시지 큐에 저장된 메시지의 개수를 확인하여 메시지 개수만큼 OSQPend() 수행.
    - 임의의 클록 수 (1-5)만큼 sleep
    - sleep을 수행한 누적 회수를 log에 함께 기록
    - 누적 회수는 최대 255회까지 기록. 256회는 0회로 reset됨
  - 일반 태스크
    - 코드를 수정하지 않는다.

코드

```
#include "includes.h"
#include <time.h>

#define  TASK_STK_SIZE                  512
#define  N_TASKS                        2
#define  N_MSG                          100

OS_STK    TaskStk[N_TASKS][TASK_STK_SIZE];
OS_STK    LogTaskStk[TASK_STK_SIZE];

OS_EVENT *msg_q;                                  // (1)
void          *msg_array[N_MSG];                  // (2)


void  LogTask(void *data);
void  Task(void *data);
void  CreateTasks(void);
```

코드

```
int  main (void)
{
     OSInit();
    CreateTasks();                                                    // (3)
    msg_q = OSQCreate(msg_array, (INT16U) N_MSG);                     // (4)

    if (msg_q == 0)
    {
            printf("creating msg_q is failed₩n");
            return -1;
    }
     OSStart();
     return 0;
}

void  CreateTasks (void)                                              // (5)
{
    OSTaskCreate(LogTask, (void *) 0, &LogTaskStk[TASK_STK_SIZE - 1], (INT8U) (0));
    OSTaskCreate(Task, (void *) 0, &TaskStk[0][TASK_STK_SIZE - 1], (INT8U) (10));
    OSTaskCreate(Task, (void *) 0, &TaskStk[1][TASK_STK_SIZE - 1], (INT8U) (20));
}
```

코드

```
void  LogTask (void *pdata)
{
    void      *msg;
    INT8U     sleep_count = 0, sleep_time, i, err;
    FILE      *log;
    OS_Q_DATA           q_data;

    srand(time((unsigned int *) 0));
    log = fopen("log.txt", "w");

    for (;;) {
            OSQQuery(msg_q, &q_data);                                   // (6)
            for (i=0; i<q_data.OSNMsgs; i++) {                          // (7)
                    msg = OSQPend(msg_q, 0, &err);
                    if (msg != 0) {
                            fprintf(log, "%3u: %s", sleep_count, msg);
                            fflush(log);
                    }
            }
            sleep_count++;                                              // (8)
            sleep_time = (rand() % 5) + 1;
            OSTimeDly(sleep_time);                                      // (9)
    }
}
```

코드

```
void  Task (void *pdata)                                        // (10)
{
    INT8U  sleep, err;
    char   msg[100];

    srand(time((unsigned int *) 0) + (OSTCBCur->OSTCBPrio));    // (11)

    for (;;) {
        sprintf(msg, "%4u: Task %u schedule\n", OSTimeGet(),
                        OSTCBCur->OSTCBPrio);                   // (12)
        err = OSQPost(msg_q, msg);                              // (13)
        while (err != OS_NO_ERR)
        {
            err = OSQPost(msg_q, msg);
        }
        sleep = (rand() % 5) + 1;
        OSTimeDly(sleep);                                       // (14)
    }
}
```