Regular paper http://dx.doi.org/10.13067/JKIECS.2013.8.6.839

# 선박관리 시스템의 최적화

Lim Chia Syan* · 박수홍**

## Optimization of Ship Management System

Lim Chia Syan* · Soo-Hong Park**

요 약

본 연구에서는 실시간 선박관리시스템을 위한 최적 프로그램방법을 설계 및 개발하였다. 종래의 인터럽트 프로그램방식을 대신하여 다중작업과 시각화되어진 임베디드 실시간 관리시스템을 제안한다. 데이터 관리는 임베디드 실시간 운영체계상에서 개발하였고, 인공지능방식으로 중앙연산장치(CPU)를 최적화하도록 설계되었다. 최종적으로 시스템에서의 데이터손실을 최소화하면서 최적프로그램모델을 통하여 데이터처리를 향상시켰다.

ABSTRACT

In this paper, an effort has been made to design and develop an optimized programming model for Real-time Ship Management System. Replacing the conventional interrupt-driven programming model, an embedded real-time operating system (RTOS) has been implemented on the system, allowing processes to run virtually simultaneous and multitasking. Data management algorithms are designed and developed in the RTOS to facilitate data distribution amongst tasks and optimize the CPU processing time through intelligent resource utilization. Finally, data lost in the system has been minimized via the improvement of data processing rate under the optimized programming model.

키워드

RTOS, FreeRTOS, Ship Management System, Programming Model, Optimization
RTOS, FreeRTOS, 선박 관리 시스템, 프로그래밍 모델, 최적화

## Ⅰ. INTRODUCTION

Dealing with heavy data processing for sensors and communication devices, resource management and distribution in a Real-time Ship Management System must be emphasized especially in an embedded system with limit processing power and resource in order to minimize data lost during possible. Previously developed interrupt-driven programming model has promoted insufficiency in data processing where the data buffer allocated is flooded with data.

The primary objective of this research is to design and optimize the system using embedded

* 동서대학교 대학원 메카트로닉스공학과(clwong88@hotmail.com)
** 교신저자(corresponding author) : 동서대학교 메카트로닉스공학과(shpark@dongseo.ac.kr)
접수일자 : 2013. 04. 26 심사(수정)일자 : 2013. 05. 20 게재확정일자 : 2013. 06. 21

real-time operating system (RTOS), replacing the conventional interrupt-driven programming model. Moreover, the optimized model is designed where minimum firmware update and modification are expected when new devices are added to the system or modification in hardware configuration. [5,6,7]

In this paper system, description is described first, displaying the configuration of the Real-time Ship Management System. Then, the software architecture where each optimized process algorithm is addressed in details. The result of comparing the conventional programming model with the RTOS system is also presented next in this paper. Finally, conclusion is briefed at the ending.[8,9,10]

## II. SYSTEM MODEL AND METHODS

The main text is recommended to be arranged in this order, if possible: Introduction, System Model and Methods, Results, Discussion and Conclusions. There are four levels of section headings with established specifications. Enumeration of section heads is required.

### *2.1. Software Architecture*

At low level system initialization, direct memory access (DMA) is utilized to assist data transfer between communication peripherals and CPU where the process is carried out on the background until the associated DMA buffer is completely filled with data and an interrupt is raised. As a result, there is no direct relation between peripherals and RTOS as any read-write process is achieved through DMA buffer and thus CPU processing time could be further optimized.

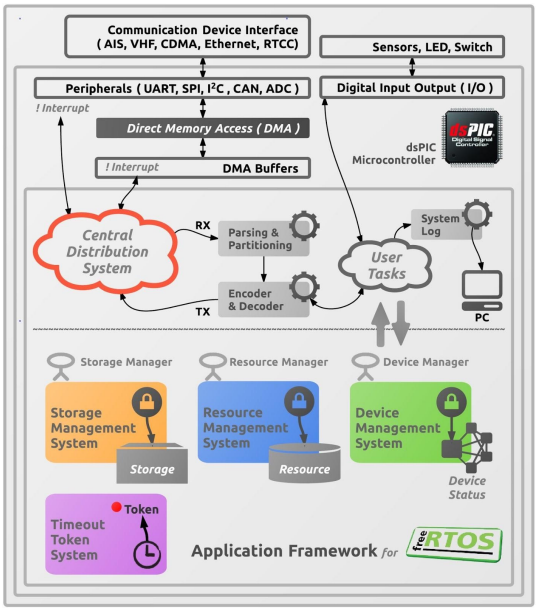



Fig. 1 Software architecture

On the other side, data management algorithms are developed and deployed on the RTOS to assist the data transfer and prevent data overloading in the system. Any error or failure detected in the system is logged in order to notify administrator for taking further actions. [1, 2] When interfacing with several devices that produce relatively large amount of data bytes, data storage becomes a crucial aspect to be considered to ensure CPU processing does not being stalled when accessing the memory. Since data bytes are received consistently from device such as AIS, a static byte array could be allocated to buffer the incoming data. Depending on the amount of data bytes received and the data bytes being completely processed, the size of the array

### *2.2. Framework Algorithm*

#### *2.2.1 Storage Management System*

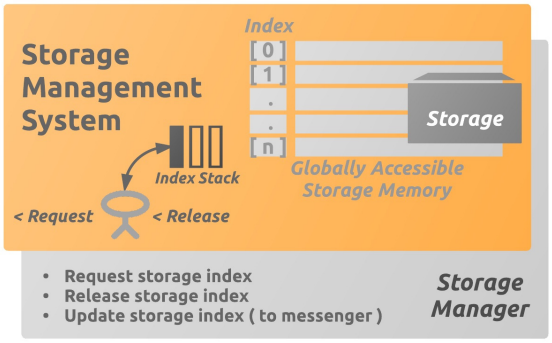



Fig. 2 Storage manager

When interfacing with several devices that produce relatively large amount of data bytes, data storage becomes a crucial aspect to be considered to ensure CPU processing does not being stalled when accessing the memory. Since data bytes are received consistently from device such as AIS, a static byte array could be allocated to buffer the incoming data. Depending on the amount of data bytes received and the data bytes being completely processed, the size of the array must be allocated adequately to hold all the incoming data. The process begins when the DMA interrupt of a peripheral is raised where data bytes arrive at the DMA buffer. [3] In the interrupt service routine, an index of available array location is requested from the empty location index queue. The moment the index is obtained, it will be removed from the queue and the particular location become unavailable. Next, the data bytes in the DMA buffer are copied into the array location via the index acquired. Lastly, the index should be sent to data ready location index queue where each queue is independent for each device attached to the system. When the data bytes are intended for processing by user task, the index is requested from the data ready location index queue and data bytes are read from the array location. Upon completion, the array location becomes available once again where the index should be sent back to the empty location index queue from the user task. This concludes the overall working principle of the Data Storage Management System. [4]

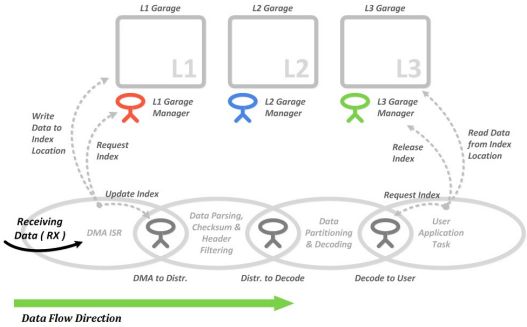



Fig. 3 Storage management system

### 1) L1 (Low Level)

When dealing with the hardware peripherals and DMA of the system, any data acquired in this low level operation should be stored in the L1 garage. In current system configuration when DMA interrupt is raised, index is requested from the L1 manager and data being hold in the DMA buffer is copied into the L1 garage using the index obtained. Lastly, index is updated into "DMA to Distr." inter-task manager before exiting the ISR without any further operations.

### 2) L2 (Intermediate Level)

In Central Distribution System, the "DMA to Distr." inter-task manager is constantly inquired to look for the index to fetch data from L1 garage. Once an index becomes available, L2 garage manager is requested and a new garage location is allocated. When data is completely parsed, the result is written to the new L2 garage location and L1 index should be released back to L1 garage manager, indicating the data in L1 garage has been fully processed.

If an incomplete data is encountered during data parsing, index of partial data should be sent to "Incomplete Data" intra-task manager. When the next set of data (contain the next fragment of

incomplete data) reaches the Central Distribution System, the "Incomplete Data" intra task manager is requested and the current fragment is combined to the index-pointed location. Finally, the index is updated to "Distr. to Decode" inter task manager, where successfully parsed data should be partitioned and decoded in the subsequence task.

### 3) L3 (High Level)

In the data partitioning and decoding task, previously parsed data is retrieved from L2 garage though "Distr. to Decode" inter-task manager. Successfully decoded data is stored in L3 garage using the similar mechanism and to be accessed by user created task.

### 2.2.2. Resource Management System

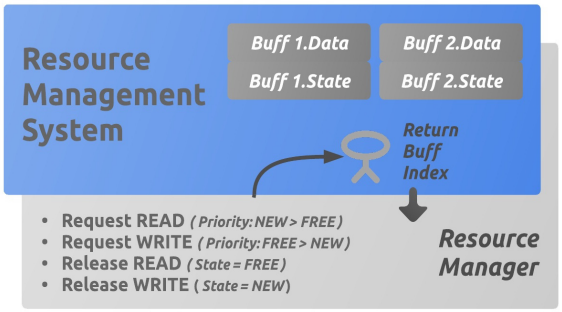



Fig. 4 Resource manager

The resource management system has been redesigned due to the performance issue on the previous version. Based on the illustration above, the resources are constantly updated regardless of the read operation by the ship information gathering task triggered by time-out, distance interval and manual invocation. In order to avoid read-write collision during gathering operation, a resource manager and a double-buffer has been developed to safeguarding the resource.

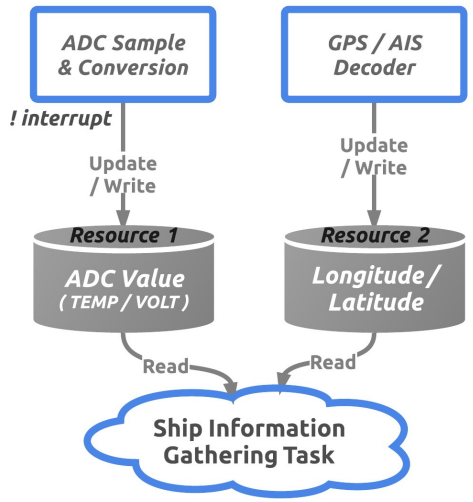



Fig. 5 Resource sharing

The working principle of the system is rather simple to understand, where a resource manager is used to interact with the two buffers based on the state. All buffers are empty and in FREE state initially since no read and write operations to the buffers exist. Depending on the operation to be performed to the resource buffer, when request is made to the resource manager, the state of the two buffers is searched to match the state required to perform the operation respectively.

For every read or write operation requested, the resource must be release through the resource manager upon operation completion, preventing buffer access deadlock.

### 2.2.3. Device Management System

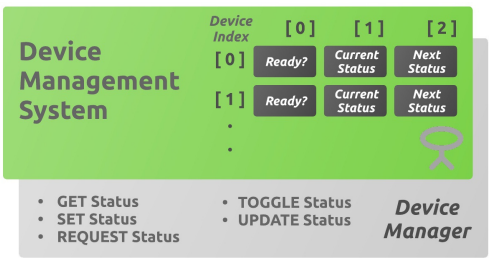



Fig. 6 Device manager

The device management system is used to organize the status for each device connected to the system, when interaction could be made through the device manager using commands supported. Since the device status must be accessible throughout the whole source code, read-write collision avoidance mechanism must be implemented especially in a RTOS environment where task could be preempted. For each device defined in the system, 3 bytes are allocated in device management system to facilitate the operation. The first byte indicates if the current device status is valid give READY or PENDING being read from the byte. In addition, if a message is intended to be sent without device specification provided, the device manager could be instructed using REQUEST command to select the ONLINE device sequentially through the device list (prioritized selection). Lastly, the device manager could be invoked to toggle the specific device status, which is more likely to be used when debugging the system.

### *2.2.4 Central Distribution System*

Often when a new device is added to the system, a considerably amount of time is required to reprogram and reconfigure the firmware of the system. Therefore, the Central Distribution System has been introduced in the system to ease the data flows between user application task and peripherals interfacing. Implementation of this solution requires only new data format to be updated in the firmware and minor change to the data encoding / decoding process.

#### *a) Transmission (TX)*

When data is intended to be sent out from the system by an application task, the data must be sent to the raw data queue with the task ID and device ID attached which is used to identify and differentiate the functionalities for each data. Thus, data for transmission across all other application tasks should be sent to the queue.

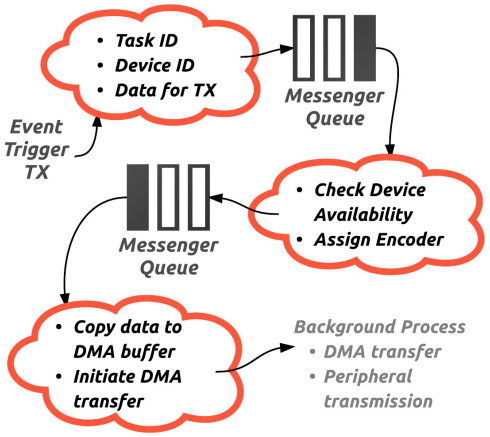



Fig. 7 Transmission using central distribution system

Upon receiving the raw data at the system, device availability is checked if the device ID has been assigned by the application, or else the device will be chosen automatically based on the task ID provided. Next, the raw data is encoded based on the selected device and copied into respective DMA buffer before the DMA transfer is initiated manually.

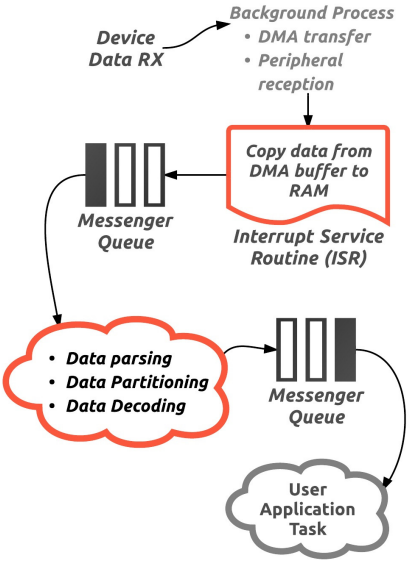



Fig. 8 Reception using central distribution system

*b) Reception (RX)*

When the data from device attached to the system reaches the peripheral, DMA transfer is initiated automatically to move the data from peripheral buffer into DMA buffer. Upon completion of transfer, interrupt is raised and data is moved to the encoded data queue along with the device ID tagged along.

The subsequence task could be categorized into three processes, extracting the information received from the device. Since the arriving data generally has different length and separated into fragments, data is parsed based on the provided data format and combine the multiple fragments data. Next, the data is searched from the beginning and partitioned when a delimiter is encountered.

Data decoding process is carried out from the parts extracted using the algorithm programmed. Lastly, the decoded data pack is tagged along with the device ID earlier before sending to the queue which could be accessed and fetched by any assigned application task.

### *2.2.5 Timeout Token System*

Data loss or corruption often takes place in an unreliable communication medium where certain data recovery algorithm must be imposed to the system to ensure data integrity. Communication reliability validation such as checksum calculation, acknowledgement system, timeout and retrans-mission are often implemented in a low cost embedded system. For example, the timeout plays a vital role to indicate communication might have failed and subsequence decision must be carried out.

However, conventional timeout programming in embedded system might cause CPU stalling because operation is performed in the interrupt service routine when timeout interrupt is raised. Moreover, timeout initiation has no standardized methodology which could be tedious sometime. Thus, the Timeout Token Request System has been designed in the system to ease the timeout request and timeout interrupt process.

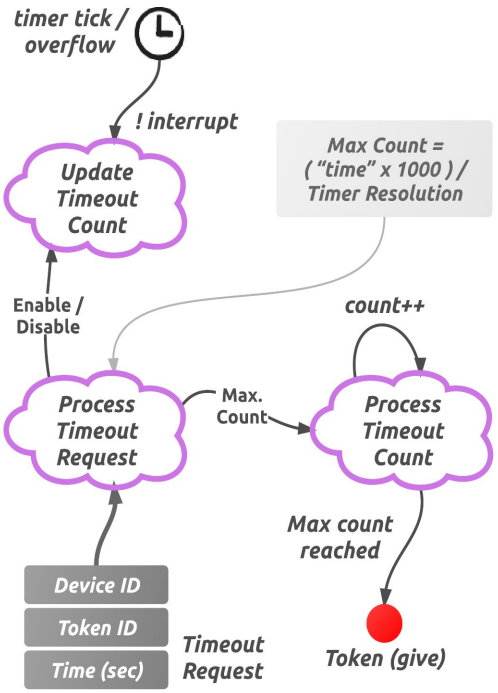



Fig. 9 Timeout token system

A hardware timer peripheral is initialized to continuously create ticks at timer resolution where the timer overflows. Since the timeout interval is relatively large compared to the timer resolution, a count is used to track the requested timeout. When a timeout token is requested by an application task, some information is required to construct a unique ID for the token.

Firstly, the device (ID) that the task intends to use as the communication medium, followed by the token (ID) which is the usage of the requested timeout. Lastly, the timeout interval required is added to the information before being sent to the timeout token info queue.

Upon receiving the information from the queue, the requested token ID is activated and the system

calculates and converts the timeout interval into the maximum timeout count which is the total numbers timer ticks to reach the desired timeout interval. For each timer tick occurs the count is incremented and compared to the maximum timeout count. Once the comparison matched, a token in the form of binary semaphore will be released by the system indicating the timeout has been hit.

Once token information is available and received from the queue, maximum count which is the counts required by the task to archive the time interval specified by the user via the timer resolution is calculated and being updated. At the same time, the token status is also switched to ACTIVE by the same process which means the count for the specific token will be incremented for each tick.

## III. RESULTS

The effort of measuring the system performance is carried out by comparing the total number of data bytes are being processed at the time interval of 500us (0.5 millisecond) between the RTOS implementation and conventional programming model in the Real-time Ship Management System.

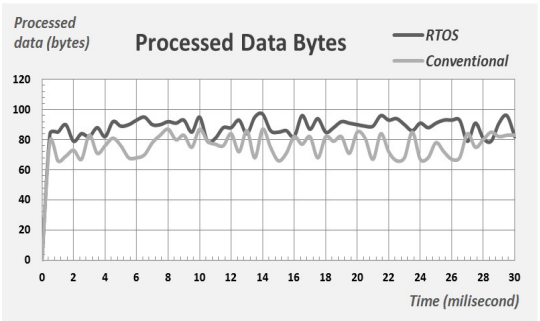



Fig. 10 Processed data bytes

Table 1. Comparison of data processing performance between RTOS and interrupt-driven programming model

| Descriptions | RTOS | conventional |
|---|---|---|
| Average data bytes processed for every 500us | 88.7 bytes | 76.4 bytes |
| Average data processing rate | 177.4 bytes/s | 152.8 bytes/s |
| percentage of improvement in data processing rate | 16% improvement | |

## IV. DISCUSSION AND CONCLUSIONS

Referring to the result illustrated in Fig. 6, it is noticeable that some improvements are achieved based on RTOS implementation in the Real-time Ship Management System. RTOS implementation adds 16% improvement of data processing rate to the current system. However, additional improvements could be achieved via further optimization on the algorithms that are currently deployed.

## REFERENCES

[1] FreeRTOS Quick Start Guide. [Online]. Available : http://www.freertos.org/FreeRTOS-quick-start-guide.html

[2] Microchip dsPIC33EP Reference Manual [Online]. Available : http://www.microchip.com/ wwwproducts/Devices.aspx?dDocName=en554307

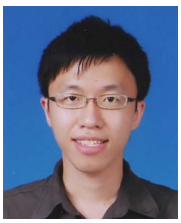
[3] Direct Memory Access (DMA) [Online].Available : http://ww1.microchip.com/downloads/en/DeviceDoc/70348C.pdf

[4] Data Memory Reference Manual [Online]. Available : http://ww1.microchip.com /downloads/en/DeviceDoc/DS-70595C.pdf

[5] Soohong Park, Lim Chia San, "Optimization of Real-Time Ship Management System using FreeRTOS", The Korea Institute of Electronic

Communication Sciences, Autumn conference, 2012.
[6] Wonbu Lee, Soohong Park,"A study on the Automatic ocean wave observation buoy system", The Journal of The Korea Institute of Electronic Communication Sciences, Vol. 6, No. 2, pp. 268-273, 2011.
[7] Hakhyun Cho, Jochun Choi, "A Study on the Navigation Data Transmission-Management System of a Small Vessel", International Journal of Maritime Information and Communication Sciences, Vol. 4, No. 1, pp. 191-200, 2000.
[8] Hyun-Sik Shin, Dong-Uk Gil, Seok-Min Yun, "A Study on Maritime Telecommunicaion of Practical Use for Pusan Area", The Journal of The Korea Institute of Electronic Communication Sciences, Vol. 1, No. 2, pp. 145-151, 2006.
[9] Woon-Taek Hwang, "A Design and Implementation of Maritime Data Communication System Focused on the GMDSS", The Journal of The Korea Institute of Electronic Communication Sciences, Vol. 6, No. 6, pp. 815-822, 2011.
[10] Wan-Pyo Hong, "Study on the ASCII Code in the side of the Transmission Efficiency in Data Communications", The Journal of The Korea Institute of Electronic Communication Sciences, Vol. 6, No. 5, pp. 657-664, 2011.

## 저자 소개



**임치산(Lim Chia Syan)**

2011년 7월 말레지아 MultiMedia University 졸업(공학사)
2013년 8월 동서대학교 대학원 메카트로닉스공학과 졸업예정
※ 관심분야 : 마이트로프로세서응용

**박수홍(Soo-Hong Park)**

1986년 2월 부산대학교 정밀기계공학과 졸업 (공학사)
1989년 2월 부산대학교 대학원 기계공학과 졸업(공학석사)
1993년 2월 부산대학교 기계공학과 졸업(공학박사)
동서대학교 메카트로닉스공학과 교수
※ 관심분야 : 제어 자동화, 로봇공학