실시간 운영체제에서 효율적인 메시지 큐 제어*

류현수[0], 이재규, 성영락[†], 이철훈
충남대학교 컴퓨터공학과, [†] 국민대학교 전자정보통신공학부
{hsryu, jklee, chlee}@ce.cnu.ac.kr, [†] yeong@mail.kookmin.ac.kr

Effective Control of Message Queues on Real-Time Operating Systems

Hyeon-Soo Ryu[0], Jae-Gyu Lee, Yeong Rak Seong[†], Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
[†] School of Electrical Engineering, Kookmin Univ.

요 약

실시간 운영체제는 여러 개의 독립적인 태스크가 동시에 실행될 수 있는 멀티태스킹 환경을 제공한다. 이러한 독립적인 태스크들 사이에 정보를 주고받기 위해서는 태스크들 사이의 통신이 필요하며 이러한 ITC(Inter-Task Communication)를 지원하는 방법에는 Global Variable 과 Message 전송 두 가지 방법이 있다. Global Variable 은 Data Corruption 의 위험이 있기 때문에 보통의 실시간 운영체제에서는 Message 전송 방식을 주로 사용하는데 이러한 방식에는 메시지 큐, 메시지 메일박스, 메시지 포트 등이 있다.

본 논문에서는 태스크들 간에 또는 태스크와 ISR(Interrupt Service Routine)이 여러 개의 메시지를 서로 전달할 수 있는 메시지 큐에 대해서 설명하고 있다. 또한 태스크 또는 ISR 이 메시지 큐로부터 메시지를 주고 받을 때 메모리의 동적 할당을 보다 효과적으로 처리함으로써 효율적인 메시지 큐 제어기법에 대해서 제시하고 있다.

1. 서론

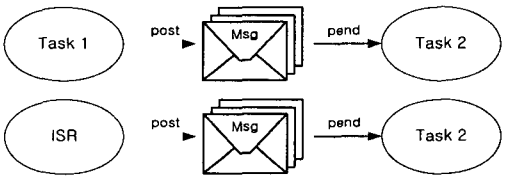
실시간 운영체제에서는 여러 개의 태스크가 동시에 실행될 수 있는 멀티태스킹 환경을 제공하는데, 태스크들 간에 정보를 주고받기 위해 ITC(Inter-Task Communication)을 제공하고 있다. 태스크와 태스크 또는 ISR(Interrupt Service Routine)과 태스크 사이에서 메시지를 전달하기 위한 방법에는 일반적으로 Shared Memory, Message Queue, Message MailBox 등이 있는데, 태스크 기반 실시간 운영체제에서는 Shared Memory 을 제공하지 않는다. 본 논문에서 기술하는 운영체제는 ITC 도구로서 Message Queue, Message MailBox 와 Message Port, Task Port 를 제공한다.

Message Queue 와 Message MailBox 는 기존의 것과 유사하지만 일부 운영체제에서는 메시지의 포인터를 전달하는데 비해 본 연구의 실시간 운영체제에서는 메시지 자체를 복사해준다. 이는 각 태스크의 메모리 영역에 대한 보호(Memory Protection)를 위한 것이다.

본 논문에서는 메시지 큐를 이용한 ITC 를 좀 더 효율적으로 적용하였다. 본 논문의 구성은 2 장에서 관련 연구를, 3 장에서는 효율적인 메시지 큐 제어를, 4 장에서는 테스트 환경 및 결과를, 5 장에서는 결론 및 향후 연구 과제에 대해 기술한다.

2. 관련 연구

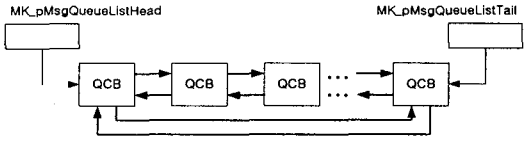
본 연구팀에서 개발한 실시간 운영체제인 iRTOS™ 커널은 메시지 큐를 통해 전달되는 메시지의 크기의 가변성에 따라 가변길이 메시지 큐와 고정길이 메시지 큐를 제공하며 포인터의 전달이 아닌 복사에 의한 전달 방법을 사용한다.



[그림 2-1] 메시지 큐

2.1 메시지 큐 리스트

iRTOS™ 커널은 메시지 큐를 동적으로 생성 및 삭제할 수 있다. 새로 생성된 메시지 큐는 메시지 큐 리스트의 가장 끝에 위치하며, 삭제 시에는 리스트에서 완전히 제거된다. 메시지 큐 리스트는 가변길이와 고정길이의 메시지 큐를 함께 관리한다.

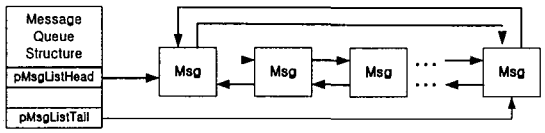


[그림 2-2] 메시지 큐 리스트

* 본 논문은 산업자원부 중기거점과제 연구비 지원에 의한 것임

2.2 메시지 큐 데이터 구조

[그림 2-3]은 메시지 큐 구조에서 pMsgListHead 와 pMsgListTail 이 메시지의 처음과 끝을 가리키고 있으며, 메시지 사이에 이중 순환 연결구조로 이루어져 있다. 메시지 큐에 메시지를 저장할 때에도 가변길이 메시지 큐와 고정길이 메시지 큐에 차이가 있다. 고정길이 메시지 큐는 내부적인 메모리 풀에서 버퍼 하나를 할당하여 메시지를 저장하고, 메시지 큐에서 태스크로 메시지를 전달하면 해당 버퍼를 메모리 풀에 반납한다. 가변길이 메시지 큐에서는 내부적인 힙 스토리지 메니저에서 필요한 메모리를 할당 받아 메시지를 저장하고, 메시지 큐에서 태스크로 메시지를 전달하면 해당 메모리를 힙 스토리지 메니저에 반납한다.



[그림 2-3] 메시지 큐 구조

[표 2-1]은 메시지 큐 제어 블록을 보여주고 있다.

필 드	설 명
m_Type	메시지 큐의 타입을 명시
m_pNext	다음 메시지 큐를 가리키는 포인터
m_pPrev	이전 메시지 큐를 가리키는 포인터
m_pName	메시지 큐의 이름
mu_FixedQueue mu_VariableQueue	고정길이 메시지 큐 가변길이 메시지 큐 (union)

[표 2-1] 메시지 큐 제어 블록

메시지 큐의 구조는 기본적으로 고정길이 메시지 큐 또는 가변길이 메시지 큐를 명시하는 m_Type 과 m_pNext, m_pPrev 와 메시지 큐의 이름인 pName 과 유니온(union)을 사용하여 고정길이 메시지 큐의 구조와 가변길이 메시지 큐의 구조를 설계하였다.

고정길이 메시지 큐는 내부적으로 메모리 풀을 사용하여 고정크기 버퍼를 할당, 해제하는 방법을 사용하고, 가변길이 메시지 큐는 내부적으로 힙 스토리지 메니저를 사용하여 가변 크기의 메모리를 할당, 해제할 수 있다. 메시지의 구조는 [표 2-2]와 같이 메시지의 시작 위치, 메시지 크기와 다음 메시지를 가리키는 포인터로 구성되어 있다.

필 드	설 명
fm(vm)_StartOfMsg	메시지의 시작 위치
fm(vm)_Length	메시지의 크기
fm(vm)_pNext	다음 메시지를 가리키는 포인터

[표 2-2] 메시지의 구조

2.3 메시지 큐의 생성과 삭제

메시지 큐를 생성하기 위해 MK_CreateQueue() 함수를 사용한다. 메시지 큐는 메시지 크기의 가변성 여부에 따라 고정길이 메시지 큐와 가변길이 메시지 큐로 구분된다. 고정길이 메시지 큐로 사용할 때에는 Type 을 MK_FIXED_SIZE 로, Size 는 최대 메시지의 개수로 설정하고, MsgSize 를 메시지의 크기로 설정하여 생성한다. 따라서 메시지 큐에 저장되는 메시지의 개수가 사용자의 설정에 따라서 제한된다. 가변길이 메시지 큐로 사용할 때에는 Type 은 MK_VARIABLE_SIZE 로, Size 는 메시지를 저장하기 위해 할당된 메모리의 크기로 설정하고, MsgSize 는 메시지의 최대 크기로 설정한다. 따라서 가변길이 메시지 큐는 저장되는 메시지의 개수를 제한하는 것이 아니라 전체 메시지를 저장할 수 있는 최대 크기를 제한한다. Options 는 태스크가 기다리는 PENDING 상태의 태스크 스케줄링 정책이다.

메시지 큐의 삭제는 MK_DeleteMsgQueue() 함수를 사용하며 인자로 메시지 큐 제어 블록을 전달한다. 그러나 삭제될 메시지 큐에서 메시지를 받거나, 삭제될 메시지 큐로 메시지를 전달하기 위해 기다리는 PENDING 상태의 태스크는 메시지 큐가 삭제되면, 절대 메시지 큐에 메시지를 전달하거나 메시지 큐로부터 메시지를 받을 수 없고 삭제된 메시지 큐에서 메시지를 전달하거나 메시지 큐로부터 메시지를 받을 때까지 무한정 기다리는 문제가 발생한다. 이 문제를 해결하기 위해 메시지 큐가 삭제될 때 메시지 큐에서 기다리는 PENDING 상태의 태스크는 모두 강제적으로 READY 상태로 변경하여 다시 스케줄링 될 수 있도록 한다.

2.4 메시지 큐 제어

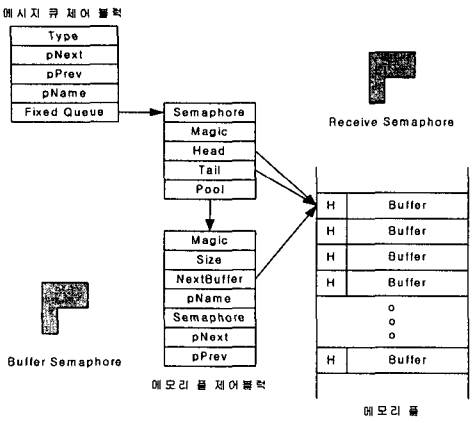
메시지 큐에서 태스크에 메시지를 전달하기 위해서 MK_MsgQueuePend() 함수를 사용한다. 인자로 pQueue 는 메시지 큐 제어 블록, pMsg 는 전달 받을 메시지, BufSize 는 메시지가 들어있는 버퍼의 크기, pLength 는 전달 받은 메시지의 실제 크기, Ticks 는 태스크가 메시지 큐에서 메시지를 전달 받기 위해 기다리는 시간 틱이다. 태스크가 메시지 큐에 메시지를 요구했을 때, 전달 가능한 메시지가 있으면 메시지를 태스크게 전달해 주고, 전달 가능한 메시지가 없으면 태스크는 메시지 큐에서 메시지를 기다리는 PENDING 상태가 되어 메시지를 전달 받기를 기다린다. 메시지 전달 받기를 기다리는 태스크의 시간 결정성 보장을 위해 Ticks 를 제공하며, Ticks 에 따라 PENDING 상태의 태스크는 기다리는 시간이 결정된다. Ticks 는 임의의 양수, MK_NO_WIAT, MK_WAIT_FOREVER 로 설정할 수 있다.

태스크 또는 ISR 에서 메시지 큐에 메시지를 전달하기 위해서 MK_MsgQueuePost() 함수를 사용한다. 인자로 pQueue 는 메시지 큐 제어 블록, pMsg 는 메시지 큐에 전달할 메시지, Length 는 메시지의 크기, Ticks 는 메시지 전달을 위해 기다리는 시간 틱이다. 태스크가 메시지 큐에 메시지를 전달하려 할 때, 메시지 큐에 메시지를 저장 가능한 버퍼나 메모리가 존재하면 메시지 큐에 메시지를 전달하고, 저장 가능한 버퍼나 메모리가 존재하지 않으면 태스크는 메시지 큐에 메시지를 전달하기 위해 기다리는 PENDING 상태가 되어 메시지 전달을 기다린다. 메시지 큐에 메시지를 전달하기 위해 기다리는 태스크의 시간 결정성 보장을 위해 Ticks 를 제공한다.

3. 효율적인 메시지 큐 제어

3.1 Create Fixed Message Queue

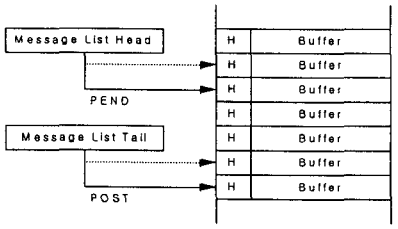
메시지 큐를 생성하면서 메모리의 모든 버퍼들을 미리 할당하여, 큐에 메시지를 저장하거나 가져올 때는 Head 와 Tail 이 해당 위치를 포인트 하도록 하였다. [그림 3-1]은 메시지 큐가 생성되면서 이루어지는 메시지 큐 제어 블록과 메모리 풀 제어 블록, 메모리 풀의 구성도를 보여주고 있다.



[그림 3-1] 메시지 큐와 메모리 풀 구성도

3.2 Fixed Message Queue Pend / Post

메시지를 큐에서 가져올 경우(PEND) 현재 Head 가 가리키는 메시지를 가져오며, Head 를 다음 버퍼로 이동시킨다. 메시지를 큐로 저장할 경우(POST) Tail 이 가리키는 위치에 메시지를 저장하고 Tail 을 다음 버퍼로 이동시킨다.

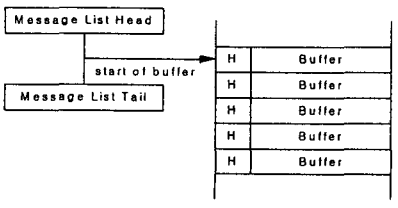


[그림 3-2] 메시지의 PEND / POST

기존의 운영체제에서 메시지를 큐에 저장하거나 가져올 때마다 메모리 할당과 해제를 했어야 했지만, 본 연구에서는 필요한 메모리영역을 미리 할당해두고 버퍼들간에 Head 와 Tail 로 순환구조를 이룸으로써 할당과 해제에 대한 오버헤드를 줄일 수 있었다.

3.3 Fixed Message Queue Reset

메시지 큐를 리셋할 경우 Head 와 Tail 이 버퍼의 시작위치를 다시 가리키게 되며, 메시지 큐를 처음 생성했을 때와 동일한 상태가 된다.

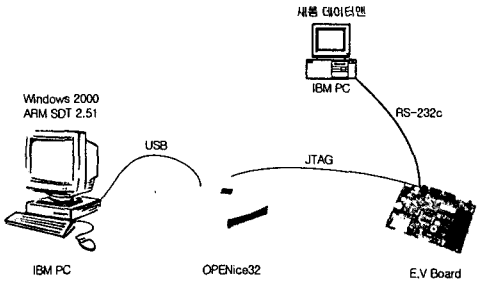


[그림 3-3] 메시지 큐 리셋

4. 테스트 환경 및 결과

위에 기술한 메시지 큐 제어는 본 연구팀에서 개발한 실시간 운영체제인 iRTOS™ 의 메시지 큐 제어에 적용하였으며, ARM920T 32-bit RICS CPU 가 탑재되어 있는 SMDK2400 보드에서 SDT2.51 통합 개발 환경을 사용하여 테스트하였다.

테스트 결과 기존의 방법보다 메시지 전달시간이 줄어듦으로써 보다 효율적인 태스크간 통신이 가능함을 확인하였다.



[그림 4-1] 테스트 환경

5. 결론 및 향후 연구 과제

본 논문에서는 멀티태스킹 환경을 제공하는 실시간 운영체제에서 ITC 의 도구인 메시지 큐의 효율적 제어에 대해서 제시하였다. 차후에 실시간 운영체제에서 태스크간 통신을 하기 위한 여러 가지 제어 기법에 대한 연구가 있어야 하겠다. 또한 효율적 메모리 관리, 파일 시스템, TCP/IP 에 대한 연구가 진행되어야 하겠다.

6. 참고문헌

[1] http://www.inestech.com
[2] David Stepner, *et. al.*, , " Embedded Application Design Using a Real-Time OS" , IEEE, 1999.
[3] Jean, J. Labrosse, *μ C/OS The Real-Time Kernel*, R&D Publications, 1995.
[4] 류철, " Embeded System 용 실시간 운영체제 설계 및 구현" , 충남대학교 석사학위논문, 2000
[5] 박희상 " 태스크기반 선점형 실시간 운영체제 설계 및 구현" 충남대학교 석사학위논문, 2003.
[6] Alan Burns & Andy Wellings, " *Real-Time Systems and Programming Languages*" , University of York, 1997
[7] Silberschatz, Galvin, Gagne, " Operating System Concept" , John Wiley & Sons, Inc. , 2003