Proceedings of the 11th
KACC, October 1996

중형항공기 시뮬레이션 소프트웨어의 작업간 공유메모리 사용의 상호배제

Mutual Exclusion of Shared Memory Access in the Simulation Software of the Midclass Commuter

°이 인석*, 이 해창**, 이 상혁**

*한국기술교육대학교 제어기계공학과 (Tel: 0417-60-1145; Fax: 0417-60-1237; E-mail: rhee@kitenms.kite.ac.kr)
**한국항공우주연구소 (Tel: 042-860-2343; Fax: 042-860-2006)

Abstracts The software of the midclass commuter flight simulation is running on multiprocessor/multitasking environment. The software is consist of tasks which are periodically alive at a given interval. Each task communicates via shared memory. The data shared by tasks is divided by several block. Only one task, called producer, can produce data for a data block but several tasks, called consumers, can read data from the data block. Double buffer and conditional flag are used to implement a mutual exclusion which prevents the producer and consumers from accessing the same data block simultaneously.

Keywords Flight simulator, Simulation, Shared memory, Mutual exclusion

1. 서론

중형항공기 시뮬레이터는 중형항공기 개발에 있어서 조종계통 설계/해석의 검증용 도구로서 개발된다. 개발 시뮬레이터는 다수의 컴퓨터가 서로 연결되어 독립적으로 작업을 수행하는 다중프로세서 컴퓨터시스템으로 구성되어 있으며 소프트웨어는 여러 개의 작업으로 구성되며, 각 작업들은 여러 프로세서에 분산하여 위치한다.[7]

공유메모리(shared memory)는 다른 프로세서에서도 엑세스가 가능하고, 엑세스 시간이 짧아서 실시간 소프트웨어에서 작업간 데이터를 서로 교환하는 방법으로 널리 사용된다. 공유메모리를 사용할 경우 여러 개의 작업이 동시에 같은 영역을 엑세스하는 경우 데이터가 손상이 될 수 있으므로 하나의 작업이외에는 다른 작업이 같은 영역을 엑세스하지 못하도록 상호배제(mutual exclusion)하여야 한다.[6] 상호배제 방법으로 semaphore가 대표적으로 사용된다[2-5]. Semaphore는 UNIX나 VxWorks, VRTX, VMEexec등 상용으로 사용되는 실시간 운영체계에서 제공하므로 쉽게 사용할 수 있다. 그러나 중형항공기 시뮬레이터에서 채택하고 있는 VMEexec에서 제공하는 semaphore를 다중프로세서에서 사용할 때에는 단일프로세서에서 사용할 때보다 속도가 매우 떨어지므로 서로 다른 프로세서내의 작업간의 데이터 통신에는 사용하기 어렵다. 참고문헌 [1]에서는 인터럽트를 사용한 상호배제방법을 사용하였다. 다중작업시스템에서 인터럽트를 사용하여 작업을 스케쥴링하므로 한 작업이 공유메모리내의 영역을 엑세스할 때는 CPU를 인터럽트 불능상태로 하여 다른 작업이 수행되는 것을 방지하고 엑세스가 끝난 후 인터럽트 가능상태로 돌리는 방법이다. 이 방법은 단일프로세서에서만 사용 가능한 방법이므로 시뮬레이터 소프트웨어에는 적용할 수가 없다. 인터럽트를 사용하는 것과 유사한 방법으로 공유메모리를 사용할 때 작업의 우선 순위를 최대로 높이는 방법[3]이 있으나 역시 단일 프로세서에서만 사용이 가능하다. 다중프로세서에서 사용할 수 있는 방법으로는 조건 flag를 사용하는 방법이 있다.[4]

앞에서 설명한 방법은 한 영역을 단지 하나의 작업만이 엑세스하도록하는 상호배제방법들이다. 시뮬레이터 소프트웨어에서 공유메모리는 몇 개의 블록으로 나뉘며 각 블록에 대해 데이터를 생성하는 생산작업은 단 하나이고 다수의 소비작업이 데이터를 사용할 수 있다. 각 작업들은 서로 다른 작업주기로 수행이 되므로 소비작업은 가장 최근에 갱신된 데이터를 읽어들여야 한다. 다수의 소비작업이 같은 영역을 엑세스할 수 있으나 생산작업과 소비작업이 동시에 같은 블록의 데이터를 엑세스하지 못하도록 공유메모리 사용의 상호배제가 이루어져야 한다. 본 연구에서는 이중버퍼를 사용하고 각 작업에 flag를 부여하는 방법을 사용하여 실시간으로 사용할 수 있는 상호배제방법을 구현하였다.

2. 시뮬레이터 소프트웨어의 구성

중형항공기 시뮬레이터의 컴퓨터 시스템은 그림 1과 같이 다중프로세서 시스템으로 구성되어 있다. Sun 워크스테이션과 호스트 컴퓨터는 Unix 운영체계를 사용하고 VMEbus에 직접 연결되어 있는 3개의 Target 컴퓨터는 Motorola사의 실시간 운영체계인 VMEexec를 사용한다. Unix 운영체계를 사용하는 컴퓨터에서는 실시간 계산이 요구되지 않는 작업이 수행되며 실시간 작업은 주로 Target 컴퓨터에서 수행된다.

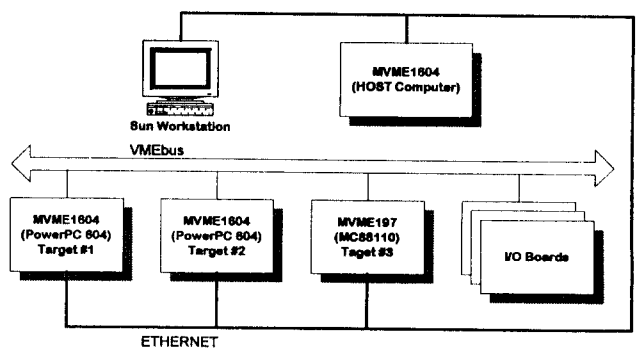

그림 1. 시뮬레이터 컴퓨터 시스템
Fig. 1. Simulator computer system

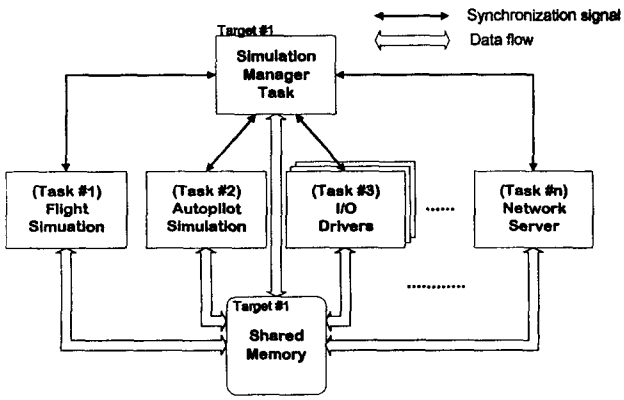

그림 2. Target 컴퓨터내의 작업 구성도
Fig. 2. Diagram of tasks in target computers

Target 컴퓨터 내에는 그림 2와 같이 여러 개의 작업이 수행된다. Simulation Manager 작업과 공유메모리를 제외한 작업은 어떠한 Target 컴퓨터에도 위치할 수 있다. 작업이 수행되는 Target 컴퓨터는 컴퓨터의 부하를 고려하여 사용자가 결정한다. Simulation Manager 작업은 실시간으로 수행되지 않으나 Target내의 작업들의 수행을 작업주기를 설정하고 동기 시키는 역할을 한다. Target #1에 설정된 공유메모리는 작업간 데이터 교환을 위해 사용한다. 작업간 공유되는 데이터는 몇 개의 블록으로 나뉜다. 각 블록의 데이터는 단 하나의 작업에 대해 생성되며, 여러 작업들이 블록의 데이터를 읽어 들일 수 있다. 즉 각 블록에 대해 하나의 생산자(producer)와 여러 개의 소비자(consumer)가 데이터를 공유한다.

3. 상호배제 프리미티브의 구현

각 블록에 대해 생산자는 일정 주기로 데이터 블록을 갱신한다. 소비자들은 수행주기가 되었을 때는 생산자가 최근에 생성한 데이터를 공유메모리로부터 읽어 들여야한다. 생산자가 데이터를 갱신하고 있는 중일 때는 데이터 블록내의 데이터는 과거의 데이터와 현재의 데이터가 공존하므로 이 때 소비자가 데이터 블록을 엑세스하면 안된다. 역으로 소비자 데이터 블록에서 데이터를 읽고 있는 중 생산자가 데이터 블록의 데이터를 갱신

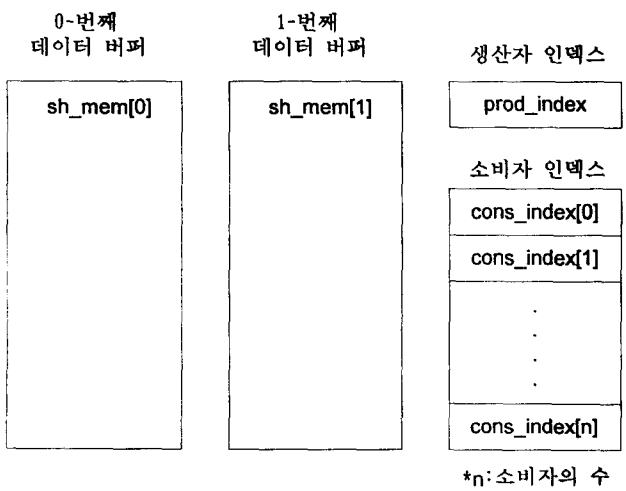

그림 3. 상호배제를 위한 공유메모리 구성
Fig. 3. Shared memory structure for mutual exclusion

하여서도 안된다. 따라서 각 블록에 대해 생산자와 소비자들간에는 동시에 같은 데이터 블록을 엑세스하지 않도록 상호배제하여야 한다. 소비자는 데이터를 변화시키지 않으므로 소비자끼리는 상호배제할 필요가 없다.

우선 하나의 데이터블록에 대해 생산자와 소비자들간의 상호배제를 고려하여보자. 이를 위해 공유메모리를 그림 3과 같이 구성한다. 공유할 데이터를 저장하기 위해 두개의 데이터 버퍼를, 생산자와 소비자가 어느 버퍼를 사용할 것인지를 나타내는 생산자 인덱스와 소비자 인덱스를 구성한다. 소비자 인덱스는 소비자수 만큼 구성한다. 생산자 인덱스는 생산자가 데이터를 현재 갱신하고 있는 또는 앞으로 갱신할 버퍼를 나타낸다. 즉 prod_index = 0이면 sh_mem[0]를 사용하고 1이면 sh_mem[1]을 사용하는 것을 나타낸다. 두개의 데이터 버퍼가 있으므로 prod_index는 0 또는 1만 될 수가 있고, 따라서, 생산자 인덱스의 1의 보수는 최근에 갱신된 데이터 버퍼를 가리킨다. 소비자 인덱스는 -1, 0, 1의 셋 중의 한 값을 가진다. -1은 해당 소비자는 버퍼의 데이터를 읽고 있지 않음을 나타내고, 0/1은 현재 소비자가 0/1번째 데이터 버퍼로부터 데이터를 읽고 있음을 나타낸다.

그림 4는 생산자와 소비자간의 상호배제 프리미티브를 나나낸다.

Producer Side:

```
Initialization of shared memory:

    Write Initial Data to sh_mem[0].
    prod_index = 1.
    for i=0 to n
        cons_index[i] = -1

Data Update:

    while LOOP for wait_flag == set
        Reset wait_flag.
        for i=0 to n
            if(prod_index == cons_index[i])
                Set wait_flag.
                Break for loop.
    End of while LOOP.

    Set task priority to the highest.
    Update data in sh_mem[prod_index].
    Take 1's complement of prod_index.
    Restore original task priority.
```

i-th Consumer Side:

```
Data Read:
    Set task priority to the highest.
    cons_index[i] = 1's comlement of prod_index

    Copy data from sh_mem[cons_index[i]]
    cons_index[i] = -1
    Restore original task priority
```

그림 4. 상호배제 프리미티브
Fig. 4. Mutual exclution primitive

생산자가 데이터를 처음 생성할 때는 0-번째 데이터 버퍼에 데이터를 쓴다. 그리고 생산자 인덱스를 1로 세팅한다. 이는 최근에 갱신된 곳은 0-번째 데이터 버퍼이고, 다음 데이터 갱신은 첫 번째 에 할 것임을 나타낸다. 생산자는 데이터를 갱신할 때 갱신할 버퍼를 소비자가 읽고 있으면 읽기가 끝날 때까지 기다린 후 갱신한다. 이는 생산자 인덱스와 소비자 인덱스가 같은지 아닌지를 조사하면 알 수 있다. 두 인덱스가 같으면 갱신할 곳의 버퍼를 소비자가 읽고 있음을 나타낸다. 데이터를 갱신하는 동안은 빠른 시간 내에 이루어져야 하므로 다른 작업으로부터 방해를 받지 않도록 생산자 작업의 우선순위를 가장 높게 올린다. 데이터 갱신이 끝나면 우선순위를 원래 값으로 돌리고, 생산자 인덱스를 1의 보수로 바꾸어 방금 갱신이 끝나 곳을 소비자에 알린다. 생산자 인덱스는 갱신이 끝나기 전에는 1의 보수로 되지 않으므로 생산자 인덱스는 현재 갱신하고 있거나 앞으로 갱신할 버퍼의 위치를 나타내게 된다.

소비자는 데이터를 읽어 들이고자 할 때는 생산자 인덱스의 1의 보수를 취해 소비자 인덱스로 하므로 최근에 데이터가 갱신된 곳으로부터 데이터를 읽어 들일 수 있고, 생산자와 소비자가 동시에 같은 메모리 블록을 엑세스하는 것이 방지된다. 소비자가 데이터를 읽어 들일 때도 다른 작업으로부터 방해를 받지 않도록 작업의 우선순위를 가장 높게 올린다.

시뮬레이션 소프트웨어에서 공유메모리는 생산자가 서로 다른 런 여러 개의 부분으로 나뉜다. 따라서 하나의 데이터 생산자가 공유메모리의 모든 부분을 사용하는 경우에 사용하는 위의 방법을 시뮬레이션작업에 적용할 수 없다. 시뮬레이션 소프트웨어에서는 공유메모리의 사용하기 위해 앞에서 제시된 방법을 그림 2와 같이 수정한다. 데이터 버퍼를 생산자에 따라 부분으로 나누고, 각 부분에 대해 인덱스를 각각 구성한다. 예를 들어 작업 1이 생산자인 부분을 Part-1, 작업 2가 생산자인 부분을 Part-2로 구분하는 것이다. 데이터의 갱신과 읽기는 위와 같은 방법을 사용하며 단지 인덱스를 읽고자 하는 부분에 해당하는 것을 사용하면 된다.

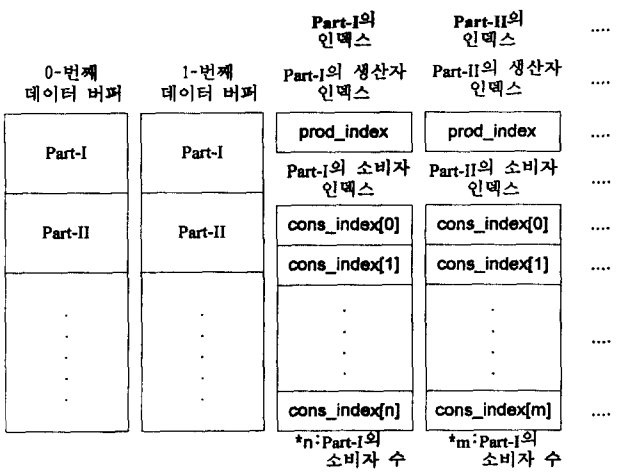

그림 4. 다수 블록으로 이루어진 데이터 버퍼의 상호배제를 위한 공유메모리 구성

Fig. 4. Shared memory structure for mutual exclusion of multi-block data buffer

4. 결론

시뮬레이션 소프트웨어의 각 작업은 주어진 주기로 수행이 되어 시뮬레이션 데이터를 생성하고, 공유메모리에 저장하여 다른 작업에 필요한 정보를 제공한다. 시뮬레이션에서 공유되는 데이터는 몇 개의 블록으로 나누어져 있으며, 각 데이터 블록에 대해 데이터를 생성하는 생산작업은 단 하나이나 데이터를 읽어들여 사용하는 소비작업은 다수이도록 소프트웨어가 설계되어 있다. 생산작업과 소비작업이 동시에 같은 블록의 데이터를 엑세스하지 못하도록 공유메모리 사용의 상호배제가 이루어져야 한다. 이를 위해 여기서는 각 데이터 블록에 대해 두 개의 메모리 버퍼를 설정하고 생산작업과 소비작업들에 인덱스를 부여한다. 생산작업은 두 개의 버퍼에 번갈아 가면서 데이터를 갱신하고 생산작업의 인덱스는 다음 주기에서 갱신할 버퍼의 위치를 가리키도록 하며, 소비작업은 현재 생산작업의 인덱스가 가리키지 않는 버퍼의 데이터를 엑세스하는 방법을 사용하여 공유메모리 사용의 상호배제를 구현하였다.

후기

본 연구는 통상산업부에서 시행한 '중형항공기 개발사업'의 연구결과의 일부입니다. 본 연구를 지원해주신 관계자 여러분께 감사드립니다.

참고문헌

[1] D. M. Auslander and C. H. Tham, *Real-Time Software for Control*, Prentice Hall, 1990.

[2] P. Laplante, *Real-Time Systems Design and Analysis*, IEEE Press, New York, 1992.

[3] Motolora, *VMEexec User's Guide*, 1994.

[4] S. Bennett, *Real-Time Computer Control: An Introduction*, Prentice Hall International, 1991.

[5] W. R. Stevens, *UNIX Network Programming*, Prentice Hall International, 1994.

[6] 조 유근, 고 건, *운영체제론*, 홍릉과학출판사, 1984.

[7] 한국항공우주연구소, *중형항공기 개발사업 연구보고서*, TR-94-201.