초경량 실시간 운영체제 TMO-eCos를 위한 TMO 지원 라이브러리 및 BCC 스케줄러

(A TMO Supporting Library and a BCC Scheduler for the Micro-scale Real-time OS, TMO-eCos)

주 현 태 † 김 정 국 ††
(Hyuntae Ju) (Jung-guk Kim)

요 약 실시간 처리의 가장 큰 목적은 시스템에서 동작하는 실시간 태스크들이 주어진 데드라인을 잘 지키도록 하는 것이다. 본 논문에서는 임베디드 시스템을 위한 운영체제인 TMO-eCos의 실시간 태스크 스케줄링 성능 개선을 위하여, TMO(Time-triggered Message-triggered Object) 모델에 필수 요소로 규정된 BCC(Basic Concurrency Control) 스케줄러의 구현과, 실시간 객체 TMO의 객체 기반 프로그래밍을 제공하는 TMO 지원 라이브러리의 설계 및 구현에 관하여 기술한다. BCC 스케줄러는 보장성 컴퓨팅 설계를 위한 것으로, 비동기적 사건 구동 태스크의 스케줄을 사전에 스케줄이 정의된 시간 구동 태스크의 구동 시간을 제외한 여유 시간이 충분할 때에만 허용하는 실시간 스케줄러이다.

키워드 : 실시간 처리, 스케줄러, TMO

Abstract It is the most important object of real-time computing to make real-time tasks keep their given time conditions. In this paper, we implemented BCC(Basic Concurrency Constraint) scheduler which is provided as an essential element of TMO(Time-triggered Message-triggered Object) model, and TMO Supporting Library that supports object-oriented design for TMO. BCC scheduler is a means to design timeliness-guaranteed computing, and it predicts the start of SpMs first, and then it makes the execution of SvMs deferred when it is predicted that any SpM begins to run currently. In this way, BCC is able to prevent collisions between SpM and SvM, and it gives higher priority to SpMs than SvMs.

Key words : Real-time computing, Scheduler, TMO

· 본 연구는 방위사업청, 국방과학연구소의 지원(UD060048AD) 및 지식경제부, 정보통신연구진흥원의 대학 IT연구센터 지원사업의(IITA-2008-C1090-0804-0015) 연구결과로 수행되었음
· 이 논문은 제35회 추계학술대회에서 '초경량 실시간 운영체제 TMO-eCos를 위한 TMO 지원 라이브러리 및 BCC 스케줄러'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : 한국외국어대학교 컴퓨터및정보통신공학과
ht420@keti.re.kr
†† 종신회원 : 한국외국어대학교 컴퓨터및정보통신공학과 교수
jgkim@hufs.ac.kr

논문접수 : 2009년 1월 19일
심사완료 : 2009년 4월 27일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제7호(2009.7)

1. 서 론

시간 보장성 컴퓨팅(time-guaranteed computing)은 설계 시 주어진 데드라인을 준수하여 작업을 처리하는 것을 의미하는데, 작업의 처리 속도보다는 데드라인 준수가 가장 중요한 목표가 된다. 이러한 보장성 컴퓨팅 설계를 위해 TMO(Time-triggered Message-triggered Object) 모델이 제안되었으며, 리눅스와 윈도우즈, eCos와 같은 다양한 운영체제를 기반으로 TMO를 지원하기 위한 미들웨어와 커널이 개발되었다.

TMO는 공유 데이터 영역인 ODS(Object Data Storage)와 주기 및 시작 시간과 같은 시간 조건에 의하여 동작하는 SpM(Spontaneous Method), 그리고 메시지의 도착에 의하여 동작하는 SvM(Service Method)으로 구성되며[1], 이들 SpM과 SvM은 주어진 데드라인을 준수하도록 스케줄링 되어야 한다. 따라서 TMO를 지원하는 미들웨어나 커널의 최대 관건은 실시간 태스크들이 얼마나 데드라인을 잘 지킬 수 있도록 스케줄링 하는가 하는 것이다.

한편, TMO 모델의 기본 요소로 제시된 BCC(Basic Concurrency Constraint)는 보장성 컴퓨팅 설계를 위한 것으로, 각 SpM의 주기와 수행시간을 바탕으로 수행 시작 시간을 예측하고, SvM 수행 전에 이를 점검하여 SpM이 수행될 것이 예상되는 시간에는 SvM의 수행을 보류하도록 함으로써 SpM과 SvM의 충돌을 방지하고, 시간 선점의 우선 순위를 사전 예측이 가능한 SpM에 주는 스케줄링 기법이다.

Klaus Schild의 논문은 두 부분으로 구성되며, 주기적 태스크의 순차화와 주기적 태스크에서 메시지를 받는 태스크의 상관관계를 고려하여 Constraint logic program으로 메시지 구동 태스크도 순차화를 하는 논문인데[2], 이런 경우는 메시지 구동 태스크의 배치에 주기적 태스크의 배치가 영향을 받게 된다[3].

이와는 달리 TMO scheme에서는 시간 보장성 컴퓨팅을 목적으로 그 동작이 사전 설계 가능한 주기적 SpM을 중시하며, 따라서 SpM을 먼저 스케줄하고 남는

시간에 SvM을 할당하게 된다. 이를 위한 스케줄링 기법이 BCC이며 본 논문에서는 이를 TMO-eCos 커널에 설계 구현하였다. 그 외의 여러 논문들은 대부분 주기적 태스크를 대상으로 한 데드라인 스케줄러이므로 TMO의 SpM과 SvM을 대상으로 하는 본 논문의 스케줄러와의 비교에 의미가 없다.

2. TMO-eCos 스케줄러

eCos 운영체제는 Bitmap, Multi-Level Queue 등의 기본 스케줄러를 제공하고 있으며, 확장 개발된 TMO-eCos에서는 Multi-Level Queue 스케줄러를 기반으로 데드라인 기반 실시간 스케줄러가 구현되었다. TMO-eCos 실시간 스케줄링의 핵심적인 역할을 담당하는 WRMT(Watchdog Real-time Management Task)는 매 클럭 인터럽트 발생 시마다 모든 실시간 태스크의 데드라인과 SpM의 주기를 계산하고 점검한다[4].

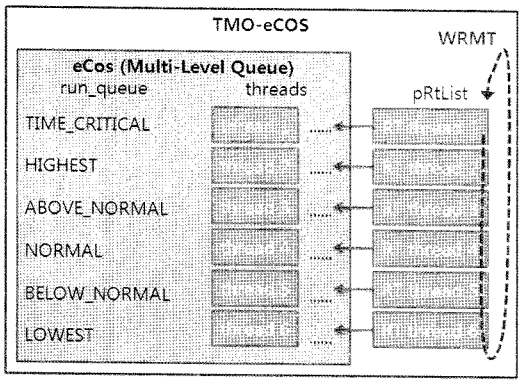

그림 1 TMO-eCos 스케줄러의 구조

TMO-eCos에서 태스크가 생성되면 eCos의 기본 스케줄러인 Multi-Level Queue의 run_queue에 등록되고 이것이 TMO의 SpM이나 SvM 실시간 스레드로 등록되면 해당 스레드는 WRMT의 TMO 태스크의 관리 리스트인 pRtList에 추가된다. TMO-eCos는 이 리스트를 토대로 LLF(Least Laxity First)와 EDF(Earliest Deadline First), FTFS(First Triggered First Served)와 같은 다양한 기법의 데드라인 기반 스케줄링 정책을 적용하고 있다.

3. BCC(Basic Concurrency Constraint)

TMO 객체 내부에 존재하는 여러 개의 실시간 태스크가 동시에 공유 데이터 영역에 접근하고자 할 때 이에 대한 중재 방법이 필요하다. 실시간 처리 환경에서의 상호배제를 위해서는 수행중인 태스크를 임의로 중단시키는 기존의 방법보다는 태스크의 중단 없이도 공유 자원에 배타적으로 접근할 수 있도록 각 태스크를 스케줄하는 방법이 적합하다[5].

SpM은 데드라인뿐만 아니라 주기에 의해 수행의 시작 시점이 엄격히 지켜져야 하는 시간 구동 태스크이므로, 보장성 컴퓨팅의 설계상 비동기적인 SvM에 우선하여 수행되도록 하는 것이 좋다. 따라서 SpM과 SvM의 충돌이 예상될 때는 SvM의 수행을 보류시키고 SpM에 우선권을 줌으로써 수행 시간 문제나 공유 자원의 충돌 문제를 해결할 수 있다. TMO의 기본 요소인 BCC 스케줄러는 시스템에 존재하는 모든 SpM의 주기를 바탕으로 수행 시작 시간을 예측하고, 또한 SvM의 수행 시마다 주어진 WCET(Worst Case Execution Time)를 토대로 SvM의 수행에 소요되는 시간을 추측한다. 이러한 시간 요소들을 토대로, SvM의 수행을 시작하기 전에 가장 수행이 임박한 SpM의 동작이 시작되기 전까지 해당 SvM의 수행을 마칠 수 있는지를 예측한다. 가장 임박한 SpM의 수행 시작 전에 해당 SvM의 수행을 완료할 수 있을 것으로 예상되면 그림 2와 같이 SvM을 스케줄하고, 그렇지 않으면 적절한 시작 시점을 찾을 때까지 그림 3과 같이 해당 SvM의 수행을 보류시키게 된다.

BCC 스케줄러의 구현을 위해 그림 1에서 제시한 TMO-eCos 스케줄러의 구조를 그림 4와 같이 확장하였다.

BCC Queue는 두 가지의 queue로 구성된다. 먼저, SpM의 수행 주기 계산을 위해 모든 SpM을 관리하는 BCC SpM 리스트를 유지한다. TMO-eCos에서 SpM으로 등록되는 메소드는 모두 pRtList와 동시에 BCC SpM 리스트에도 등록된다. WRMT에서는 매 클럭 인터럽트 발생 시마다 각 SpM의 수행 시작 시간을 계산하게 된다. 그리고 이 수행 시작 시간의 순서대로 BCC SpM List가 구성되는데, WRMT에서는 각 SpM의 수

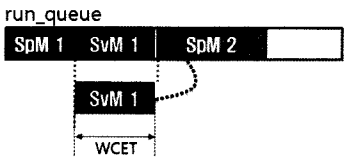

그림 2 SvM을 즉시 수행할 수 있는 경우

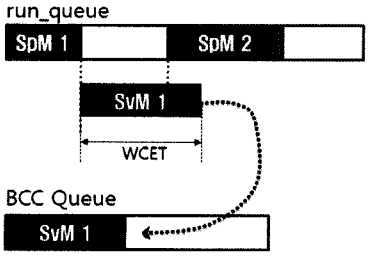

그림 3 SvM의 수행을 보류하는 경우

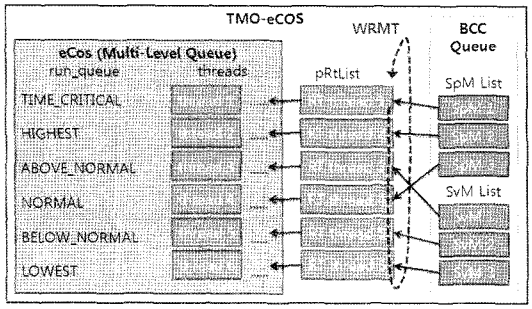

그림 4 BCC 스케줄러의 구조

행이 종료될 때 해당 SpM을 BCC SpM List의 적절한 위치로 재배치하는 작업을 해 줌으로써 리스트를 SpM의 수행 시작 시간의 순서대로 유지하게 된다. 따라서 수행이 가장 임박한 SpM은 이 BCC SpM List의 첫 번째 위치에 있는 SpM이다.

SvM은 앞에서 제시한 스케줄링 정책에 따라 수행이 보류될 경우 BCC SvM 리스트에 등록된다. SvM이 구동을 위해 사건 메시지를 기다리는 SvM_wait_invocation()은 메시지 도착시 SvM을 스케줄링 가능한 상태로 전환하기 전에 BCC 스케줄링 정책에 따른 SvM 수행 조건들을 판단하고 해당 SvM을 수행할지 혹은 BCC SvM 리스트에 등록하고 수행을 보류하도록 할지를 결정하게 된다. BCC SvM 리스트는 등록되는 순서대로 그 순서가 유지되며, 스케줄링 정책에 따라 수행이 시작된 SvM은 리스트에서 제거된다. 단, 발생된 메시지는 순서대로 처리되어야 하므로 BCC SvM 리스트에 대기중인 SvM은 리스트 등록된 순서대로 수행되어야 한다.

메시지를 받은 SvM은 BCC 스케줄링 정책에 따라 다음 세 가지 조건이 모두 만족될 때 수행을 시작할 수 있으며, 하나라도 만족하지 못하면 해당 SvM은 BCC SvM 리스트에 등록되어 대기하게 된다.

1) 수행이 가장 임박한 SpM의 시작 시간 > SvM의 WCET
2) 현 시점에 수행 중인 SpM이 없을 때
3) BCC SvM 리스트에 이미 대기중인 SvM이 없을 때

BCC를 사용하지 않는 경우에, SpM이나 SvM이 수행을 마칠 때에는 실시간 태스크의 상태가 ACTIVE에서 INACTIVE로 전환됨과 동시에 스레드의 상태는 SUSPENDED가 되는데, 이 과정은 eCos의 커널 함수 suspend()에 의해 수행된다.

본 논문에서는 BCC 확장을 위해, 하나의 실시간 태스크가 한 주기의 수행을 마치고 대기 상태로 들어가기 전에 BCC SvM List에 대기 중인 SvM이 현 시점에 수행 가능한지를 확인할 수 있도록 suspend()의 기능을 확장한 bcc_suspend()를 구현하였다. BCC SvM 리스트에 대기중인 SvM이 없으면 bcc_suspend()는 suspend()와 동일하게 동작한다. 그러나 반대의 경우에는 리스트의 가장 앞에 있는 SvM에 대하여 BCC 정책에 따른 수행 조건을 판단하고 수행 재개 여부를 결정하게 된다.

BCC 스케줄러는 시스템 구성시의 선택 사양으로, LLF나 EDF와 같은 기존의 스케줄러와 병행하여 사용할 수 있다.

4. 스케줄링 성능 분석 실험

이 실험의 목적은 BCC 스케줄링 정책을 적용하였을 때 제한된 조건의 실험에서 스케줄링 성능이 얼마나 개선되었는지를 분석하는 것으로, SpM과 SvM이 하나씩 있는 객체의 개수를 늘려 가면서 실시간 태스크의 정시 보장 성능과 SpM의 주기 보장 성능에 관한 실험을 하였다.

실험을 위한 각 메소드의 작업 내용은 다음과 같으며, SpM의 경우는 SvM의 구동을 위한 메시지 송신 작업을 포함한다.

```
unsigned int sum;

for(int a = 1; a <=0xFFFFFF; ++a)
{
        ++sum;
}
// SpM: Send a message to the SvM
```

실험에 사용된 CPU-bound 작업의 단일 수행 시 걸린 시간을 I(Individual elapsed time)라는 상수로 두고, 약간의 변화를 위해 난수를 발생시켜 R이라는 상수로 둔다. 그리고 동시에 수행되는 스레드의 개수를 N으로 두었을 때, 각 태스크에 부여되는 시간 조건은 다음과 같다.

- *SpM과 SvM의 데드라인: $D = ((I_{SpM} + R) * N_{SpM}) + ((I_{SvM} + R) * N_{SvM})$*
- *SpM의 주기: $P = D * 2$*

위의 데드라인은 공정한 테스트를 위하여 모든 태스크들이 가급적 데드라인 안에 종료될 수 있도록 하되 최소 수행 시간에 약간의 변화를 주어 경우에 따라 데드라인을 위반할 수도 있도록 만든 수식이며, 주기는 수행 시간이 데드라인을 위반한 상황에서 다음 주기의 시작이 지장을 받지 않도록 설정한 것이다. 상수 I의 값은 시스템 환경에 따라 다르지만 본 실험 환경에서 I_{SpM}은 약 130ms, I_{SvM}은 약 110ms로 측정되었다.

그림 5와 그림 6은 전체 스레드의 개수에 따라 데드라인과 주기를 위반하는 스레드의 개수를 나타낸 것이고, 그림 7은 전체 수행 시간 동안 SpM의 주기가 얼마나 지연되었는지를 나타낸 그래프이다. 데드라인의 경우는 위반을 하더라도 CPU에 큰 부하가 걸리는 상황이

아니라면 다음 주기의 수행에는 큰 영향을 주지 않는다. 그러나 다음 수행 주기는 현 시작 시점을 기준으로 계산되므로 주기 지연에 관해서는 시간을 고려하여 성능 평가를 할 필요가 있다.

BCC는 기본적으로 SvM보다 SpM에 수행의 우선권을 주는 스케줄링 방식이므로 BCC 적용 후에 SpM 주기와 관련된 성능은 확연히 개선되는 것을 볼 수 있지

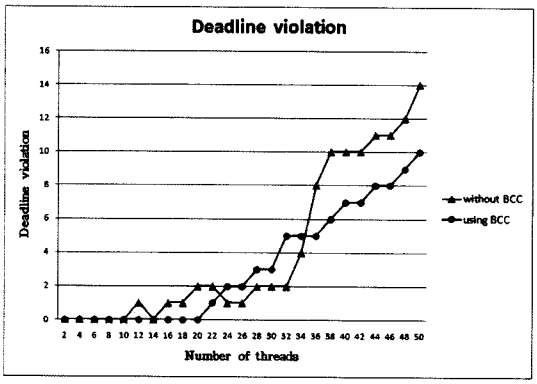

그림 5 스레드 수에 따른 데드라인 위반

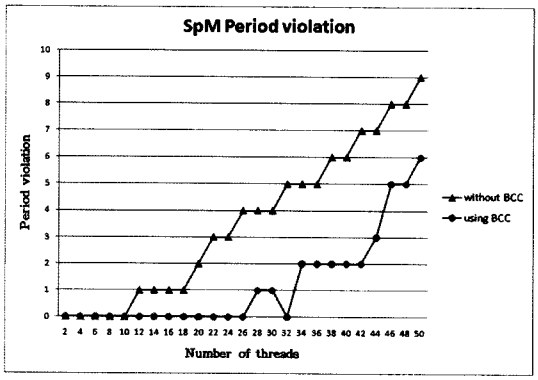

그림 6 스레드 수에 따른 SpM의 주기 위반

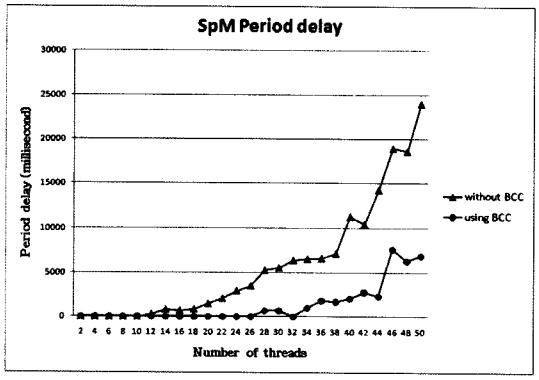

그림 7 스레드의 수에 따른 SpM 주기 지연 시간

만, 데드라인 준수율에서는 약간 특이한 점을 발견할 수 있다. BCC 적용 후에는 수행이 지연된 SvM의 데드라인 위반으로 인하여 특정 실험 구간에서는 BCC 적용 전에 비해 오히려 데드라인을 위반하는 경우가 많아진다. 그러나 태스크의 개수가 늘어나게 되면 BCC 적용 전에는 SpM과 SvM이 한꺼번에 데드라인을 위반하게 되므로 데드라인을 위반하는 경우가 기하 급수적으로 늘어나는데 비하여 BCC 적용 후에는 수행이 지연되는 SvM만 데드라인을 위반하게 되므로 BCC를 적용하기 전에 비하여 데드라인 위반 증가율이 낮아지는 것을 관찰할 수 있다.

5. TMO 프로그래밍을 위한 라이브러리

앞서 소개한 바와 같이 TMO는 공유 데이터 영역과 실시간 멤버 스레드를 포함하고 있는 객체(object)의 개념이지만[6], 기존 TMO-eCos에는 데드라인 기반의 실시간 스케줄러와 실시간 태스크 관련 API만 있을 뿐, 객체 기반으로 TMO를 구성하기 위한 도구는 마련되어 있지 않다. 따라서 지금까지 TMO-eCos 기반의 프로그래밍에서는, 객체에 소속되지 않은 시간 구동(SpM) 및 메시지 구동 태스크(SvM)를 이용하여 실시간 프로그램을 작성하였다. 본 논문에서는 이를 개선하기 위하여 용이하게 TMO의 구성을 지원하는 라이브러리를 구현하였다.

TMO 지원 라이브러리는 우선 다음과 같이 기본적인 TMO 기저 클래스를 제공하고, 사용자가 이를 상속하여 구현하고자 하는 TMO를 구성할 수 있도록 하고 있다.

```
class TMO
{
public:
  TMO();
  virtual ~TMO() { };
  virtual int InitInstance() {return 0;}
  void RegisterMethod(cyg_handle_t h);
  void start();

private:
  handleNode *handleList;
};
```

InitInstance()는 실시간 멤버 스레드의 생성과 동작에 필요한 스택 설정 및 스레드 생성을 하고, RegisterMethod() 메소드는 실시간 스레드를 해당 객체에서 일괄적으로 관리할 수 있도록 객체 내부에 리스트를 생성한다. InitInstance()는 하위 클래스에서 오버라이드 하여야 한다. start()는 RegisterMethod()에서 생성한 리스트의 메소드를 실시간 스레드로서 스케줄링을 받을 수 있는 상태로 전환하여 주는 작업을 한다.

TMO 클래스 내부에서 실시간 메소드의 선언 이후에

는 SpM()과 SvM() 매크로를 이용하여 TMO의 멤버로 선언해준다. 또한 SpM_Init() 및 SvM_Init() 매크로에 해당 메소드의 이름을 매개 변수로 주어 스택과 스레드를 생성하도록 InitInstance()를 재정의(override) 하여야 한다.

SpM이나 SvM의 본체는 다음과 같이 정의할 수 있다.

```
SpM_Body(SampleTMO, producer)
void SampleTMO::producer()
{
  SpM_Register(150, 250, 0, 0);
  AllocChannel(3, sizeof(int), 2);

  while(SpM_WaitInvocation())
  {
    // 세부 처리 내용은 생략
    SendMessage(&data, 3);
  }
}
```

eCos의 API를 사용하기 위해서는 각 메소드에 대한 주소가 필요한데, 객체의 멤버 스레드에 대하여 직접 함수 포인터 형태로 참조할 수 없으므로 이를 간접적으로 구현하기 위해 SpM_Body()와 SvM_Body() 매크로를 다음과 같이 정의하였다.

```
#define SpM_Body(ClassName, MethodName) ₩
void MethodName##_stub(cyg_addrword_t pThis)₩
{₩
  ClassName *pTMO = (ClassName *) pThis; ₩
  pTMO -> MethodName(); ₩
}
```

생성된 TMO 객체를 관리하기 위하여 TMO_Agent를 구현하였다. TMO_Agent는 TMO 지원 라이브러리 사용시 자동으로 생성되는 객체이며, 각 TMO 객체는 생성될 때 자신을 TMO_Agent에 등록하도록 생성자(constructor)가 정의되어 있다. 따라서 TMO 객체 생성 후에 TMO_Agent 객체의 begin() 메소드를 호출하여 모든 TMO 객체들이 스케줄링을 받을 수 있는 상태로 만들어 줄 수 있다.

6. 결 론

본 논문에서는 분산 실시간 객체 TMO를 지원하는 TMO-eCos 운영체제에서 SpM과 SvM의 수행 시점을 중재하여 공유 데이터 접근에 대한 충돌을 방지하고, 실시간 처리의 핵심인 데드라인 준수에 기여하도록 BCC 스케줄러의 설계와 구현에 관하여 기술하였다. 또한 TMO 객체 모델의 형틀을 제공하는 라이브러리를 구현하고 이를 활용하여 기존의 TMO-eCos에서 다소 미흡하였던, 원래의 TMO 개념에 맞는 실시간 객체 설계 방식을 제공하는 것에 관하여서도 기술하였다.

BCC는 데드라인만 가지는 SvM에 비해, 주기라고 하는 시간 제약 조건을 하나 더 가지며 수행에 대하여 사전 예측이 가능한 SpM에 수행의 우선권을 주어, 사전 설계 시부터의 보장성 컴퓨팅을 지원하는 스케줄링 기법이다.

이러한 BCC 스케줄러는 시스템 구성 시의 선택 사양으로, SpM이나 SvM의 차별을 두지 않고 단순한 데드라인 기반의 시스템을 구성할 때에는 기존의 EDF나 LLF스케줄러만 사용할 수도 있다. 따라서 BCC는 연성보다는 경성 실시간 시스템에 적합한 스케줄링 기법이라 할 수 있다.

참 고 문 헌

[1] Jung-guk Kim, Moon Hae Kim, Kwang Kim, Shin Heu, "An eCos-based Real-time Micro Operating System Supporting Execution of a TMO Structured Program," ISORC 2005, May 2005.

[2] Klaus Schild, Jorg Wurtz, "Off-Line Scheduling of a Real-Time System," *ACM Symposium on Applied Computing*, 1998.

[3] Sheng-Tzong Cheng, Ashok A. Agrawala, "Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints," Second International Workshop on Real-Time Computing Systems and Applications, IEEE, 1995.

[4] Kwang Kim, "TMO-eCos : An eCos-Based Micro Operating System Supporting A Distributed Real-Time Object Model," Doctoral dissertation, Hanyang University, 2006 (in Korean).

[5] Xuefeng Piao, Sangchul Han, Heeheon Kim, Yookun Cho, Seongje Cho, Minkyu Park, "A Non-blocking Scheme for Pre-Scheduling in Real-Time Systems," *Proc. of the KIISE Korea Fall Conference 2006*, vol.33, no.2, pp.162-166, 2006 (in Korean).

[6] K. H. Kim, "Object structures for real-time systems and simulators," *IEEE Computer*, 8, 1997.