韓國 컴퓨터情報學會 論文誌
第12卷 第5號, 2007. 11.

2007-12-5-1-12

uC/OS-II 실시간 커널의 가상화를 위한 하이퍼바이저 구현

신 동 하*, 김 지 연**

Implementation of Hypervisor for Virtualizing uC/OS-II Real Time Kernel

Dongha Shin*, Jiyeon Kim**

요 약

본 논문은 uC/OS-II 실시간 커널이 관리하는 주 자원인 마이크로프로세서와 메모리를 가상화하여 하나의 마이크로프로세서 상에서 다수의 uC/OS-II 실시간 커널을 수행시키는 하이퍼바이저를 구현하였다. 마이크로프로세서는 uC/OS-II 실시간 커널이 처리하는 인터럽트들을 제어하는 알고리즘을 적용하여 가상화하고 메모리는 물리적 메모리를 파티션하는 방식을 사용하여 가상화한다. 개발된 하이퍼바이저 프로그램은 타이머 인터럽트와 소프트웨어 인터럽트를 가상화하는 인터럽트 제어 루틴들, 하이퍼바이저와 각 커널을 정상 수행 상태까지 유도하는 코드, 그리고 가상화된 두 커널 사이에 데이터 전달을 제공하는 API로 구성되어 있다. 기존의 uC/OS-II 실시간 커널은 개발한 하이퍼바이저 상에서 수행되기 위하여 소스 코드 레벨에서 수정이 필요하다. 구현된 하이퍼바이저는 Jupiter 32비트 EISC 마이크로프로세서 상에서 실시간 동작 시험 및 독립 수행 환경 시험을 거친 결과 가상화 커널이 정상적으로 수행되는 것을 확인하였다. 본 연구 결과는 다수의 내장형 마이크로프로세서가 요구되는 응용 분야에 활용될 경우 하드웨어 가격 절감 효과를 얻을 수 있으며 내장형 시스템의 부피, 무게 및 전력 소비량을 줄이는 효과가 있음을 확인하였다.

Abstract

In this paper, we implement a hypervisor that runs multiple uC/OS-II real-time kernels on one microprocessor. The hypervisor virtualizes microprocessor and memory that are main resources managed by uC/OS-II kernel. Microprocessor is virtualized by controlling interrupts that uC/OS-II real-time kernel handles and memory is virtualized by partitioning physical memory. The hypervisor consists of three components: interrupt control routines that virtualize timer interrupt and software interrupt, a startup code that initializes the hypervisor and uC/OS-II kernels, and an API that provides communication between two kernels. The original uC/OS-II kernel needs to be modified slightly in source-code level to run on the hypervisor. We performed a real-time test and an independent computation test on Jupiter 32-bit EISC microprocessor and showed that the virtualized kernels run without problem. The result of our research can reduce the hardware cost, the system space and weight, and system power

• 제1저자 : 신동하
• 접수일 : 2007. 10.6, 심사일 : 2007. 10.19, 심사완료일 : 2007. 10.20.
* 상명대학교 소프트웨어학부 부교수 ** 상명대학교 대학원 컴퓨터과학과 석사 과정
※ 본 논문은 2007학년도 상명대학교 교내연구비에 의하여 지원되었습니다.

consumption when the hypervisor is applied in embedded applications that require many embedded microprocessors.

▸ Keyword : 가상화(Virtualization), 하이퍼바이저(Hypervisor), 내장형 시스템(Embedded System), 실시간 커널(Real-time kernel), uC/OS-II, Jupiter 32비트 EISC 마이크로프로세서 (Jupiter 32bit EISC microprocessor)

I. 서 론

본 논문은 내장형 시스템 개발에 사용되는 실시간 커널(real-time kernel)을 가상화하는 하이퍼바이저(hypervisor)의 구현에 관한 논문이다. 가상화란 컴퓨팅 자원을 나누어(divide) 사용하거나, 합쳐서(integrate) 사용하거나 혹은 시뮬레이션하는(simulate) 기술을 적용하여 실제와 다른 컴퓨팅 환경을 제공하는 기술이다[1][2]. 예를 들어 하나의 마이크로프로세서 상에서 여러 개의 프로그램을 동시에 수행시키는 멀티태스킹 기술[3], 물리적 메모리 크기와 상관없이 메모리를 구성하여 사용하는 가상 메모리 기술[4], 여러 하드 디스크를 연결하여 하나의 파일 시스템으로 구성하는 기술[5], 그리고 x86 마이크로프로세서 상에서 Alpha 마이크로프로세서를 에뮬레이션 하는 기술[6] 등이 대표적인 가상화 기술이다. 일반적으로 가상화 기술을 사용하면 컴퓨팅 자원의 활용도(utilization)를 높일 수 있고, 유연한(flexible) 컴퓨팅 자원을 제공할 수도 있으며, 시스템의 관리 및 보안 등에도 많은 이점을 제공한다[7].

본 논문에서 개발하는 하이퍼바이저는 하드웨어와 운영체제 사이에서 하드웨어 자원을 가상화하여 하나의 마이크로프로세서 상에서 다수의 운영체제를 동시에 수행시키는 소프트웨어이다[8][9]. 본 논문의 하이퍼바이저는 uC/OS-II 실시간 커널[10][11]을 대상으로 개발하는데 이 커널이 관리하는 주 자원인 마이크로프로세서와 메모리를 가상화하여 하나의 마이크로프로세서 상에서 2개 이상의 uC/OS-II 실시간 커널을 동시에 수행시킨다. 지금까지 가상화 분야 연구의 대부분이 서버 컴퓨터의 가상화[12] 혹은 데스크 탑 컴퓨터의 가상화[13]를 다루고 있지만 본 논문에서는 다수의 마이크로프로세서가 사용되는 내장형 실시간 응용 분야에 가상화 기술을 활용할 경우 하드웨어 가격 절감 효과를 얻을 수 있으며 내장형 시스템의 부피, 무게 및 전력 소비량을 줄이는 효과가 있고, 이를 실험을 통하여 확인하였다. 본 연구는 uC/OS-II 실시간 커널의 가상화에 대한 첫 시도라고 판단되며 본 연구를 시작으로 보다 개선된 방법도 제시될 수 있을 것으로 생각된다.

본 논문의 구성은 다음과 같다. 본 논문의 2절 "uC/OS-II 사전 기술" 부분에서는 가상화를 위한 하이퍼바이저 개발에 필요한 uC/OS-II 기술을 설명한다. 이 절에서는 하이퍼바이저를 설계하며 설계한 하이퍼바이저에 uC/OS-II 실시간 커널을 수정하여 포팅하는 데 필요한 기술을 설명한다. 이 절에서 설명된 기술은 본 논문 전반에서 활용될 예정이다. 3절 "하이퍼바이저 설계" 부분에서는 본 연구에서 개발하는 하이퍼바이저의 구조 및 하이퍼바이저를 구성하는 각 구성 요소의 알고리즘을 흐름도(flow chart)를 사용하여 기술한다. 개발하는 하이퍼바이저는 크게 각 커널의 인터럽트 처리 루틴을 가상화하여 제어하는 실제 인터럽트 제어 루틴, 하이퍼바이저와 각 커널을 정상 수행 상태까지 유도하는 코드, 각 커널 사이 데이터를 전달하는 API(Application Programming Interface)로 구성된다. 4절 "uC/OS-II 포팅" 부분에서는 개발하는 하이퍼바이저 상에 가상화하기 전에 수행되던 uC/OS-II를 수정하여 포팅하는 방법에 대하여 기술한다. 본 포팅은 매우 간단하여서 기존 수행 커널에 조금의 수정을 통하여 하이퍼바이저에 포팅할 수 있다. 5절 "구현 및 시험" 부분에서는 본 연구에서 사용한 내장형 마이크로프로세서인 Jupiter 32비트 EISC[14][15] 상에서 실제 구현한 방법을 설명하고, 이 마이크로프로세서 상에서 3개의 uC/OS-II 실시간 커널을 동시에 수행시킨 실험 결과를 보인다. 마지막 6절 "결론" 부분에서는 본 연구의 의의에 대하여 설명하고 앞으로 추가로 더 진행되어야 하는 연구 문제에 대하여 기술한다.

II. uC/OS-II 사전 기술

본 절에서는 하이퍼바이저 개발에 꼭 필요한 uC/OS-II 실시간 커널의 사전 기술에 대하여 설명한다.

2.1 uC/OS-II 기능 및 특징

uC/OS-II 실시간 커널은 Jean J. Labrosse가 개발한 내장형 시스템용 선점형(preemptive) 실시간 커널로서 태스크 관리, 시간 관리, 이벤트 관리(세마포, 뮤텍스, 이벤트 플래그, 메일 박스, 메시지 큐 등) 및 메모리 관리 기능을

기본 기능으로 제공한다[10][11]. 현재 버전 2.83이 발표되었으며 대학이 연구용으로 사용할 경우 라이선스 없이 소스와 오브젝트를 무료로 사용할 수 있다. 현재 소스 코드는 약 8천 라인의 C 언어로 작성되어 있고 주어진 마이크로프로세서에 포팅하기 위하여 약간의 어셈블리 언어 프로그래밍이 필요하다. uC/OS-II 실시간 커널은 자동차, 전화기, 휴대용단말기, MP3 플레이어 등과 같은 다양한 내장형 시스템에서 사용되고 있다.

2.2 uC/OS-II 메모리 모습

내장형 시스템에서 단일 uC/OS-II 실시간 커널이 여러 태스크를 서비스할 때 메모리의 모습은 그림 1과 같다. ROM은 마이크로프로세서의 인터럽트 벡터가 저장되어야 하는 0번지에서 시작되고 RAM은 그 이후 마이크로프로세서가 허용하는 번지에서 시작된다. ROM에는 보통 0번지부터 인터럽트 벡터가 저장되고 그 이후에 마이크로프로세서 및 사용 디바이스를 초기화하고 커널을 RAM으로 복사한 후 커널의 첫 번지로 제어를 넘기는 부트 로더(boot loader) 프로그램이 차지한다[14][15]. 커널의 이미지가 RAM으로 복사되기 전에는 ROM에 저장되어 있다. RAM에 저장되는 uC/OS-II 커널 이미지에는 커널이 사용하는 인터럽트 서비스 루틴이 저장되고 uC/OS-II 실시간 커널과 태스크들이 컴파일되어 저장되어있다. 또한 마이크로프로세서는 reset 인터럽트 후 부트 로더에서부터 수행이 시작되어 커널이 수행할 때 함수의 지역 변수 저장을 위하여 스택이 필요한데 이는 보통 RAM의 가장 높은 번지에서 시작하여 주소가 감소하는 방향으로 증가되며 사용된다.

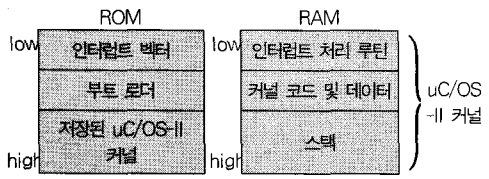

그림 1. 단일uC/OS-II 실시간 커널이 수행되는 메모리
Figure 1. Memory for single uC/OS-II kernel

2.3 uC/OS-II 인터럽트 처리 루틴

uC/OS-II 실시간 커널이 동작되기 위하여 사용하는 인터럽트는 타이머 인터럽트와 소프트웨어 인터럽트이다. 타이머 인터럽트는 주어진 시간(이 시간을 1 tick이라고 하고 하나의 커널이 수행되는 경우 보통 1/100 초로 설정되어 있음)을 주기로 발생하게끔 설정되어 있다. uC/OS-II의 타이머 인터럽트의 처리 루틴인 OSTickISR은 그림 2와 같은 일을 수행한다. 여기서 OSTickISR는 레지스터를 다루기 때문에 어셈블리 언어로 작성되어 있고 함수OSTimeTick은 C 언어로 작성되어 있다.

```
void OSTickISR(void)
{
  프로세서 레지스터를 스택에 저장한다.
  함수 OSTimeTick을 부른다.
  스택에 저장된 프로세서 레지스터를 복구한다.
}

void OSTimeTick(void)
{
  변수 t가 태스크 리스트를 가리키게 한다.
  while(t 가 마지막 태스크가 아닐 때까지) {
    t가 가리키는 태스크의 delay tick 수를 1 감소시킨다.
    변수 t가 다음 태스크를 가리키게 한다.
  }
}
```

그림 2. 타이머 인터럽트 처리 루틴
Figure 2. Timer interrupt handling routine

uC/OS-II 실시간 커널의 타이머 인터럽트 루틴인 OSTickISR이 하는 일은 각 태스크의 TCB(Task Control Block)를 점검하여 그 태스크의 delay tick 수가 설정되어 있으면 이를 1 감소시키는 일을 한다. 이 일은 실시간 처리에 있어서 중요한 일로 시간이 1 tick이 지났기 때문에 주어진 시간이 지나기를 기다리는 태스크의 기다리는 시간을 1 tick 감소시키는 일이다. 소프트웨어 인터럽트는 이 인터럽트를 발생시키는 명령어(보통 SWI 명령어)를 실행하면 발생된다. uC/OS-II에서 이 인터럽트는 어떤 태스크가 마이크로프로세서를 사용하여 수행하다가 커널이 제공하는 함수 중 커널 자원을 기다리는 함수(예: OSSemPend, OSTimeDly 등)를 부르게 되면 이를 부른 태스크는 실행 상태(running)에서 대기 상태(waiting)로 바꾸고, 현재 준비 상태(ready)에 있는 태스크들 중 가장 우선순위가 높은 태스크에게 마이크로프로세서를 할당하는 일을 한다. 이 일을 문맥 전환(context switching)이라고 부른다. 그림 3은 소프트웨어 인터럽트 처리 루틴이다. 이 루틴도 어셈블리 언어로 작성한다.

```
void OSCtxSw(void)
{
  프로세서 레지스터를 스택에 저장한다.
  OSTCBCur->OSTCBStkPtr = SP
  OSTCBCur = OSTCBHighRdy
  SP = OSTCBHighRdy->OSTCBStkPtr
  스택에 저장된 프로세서 레지스터를 복구한다.
}
```

그림 3. 소프트웨어 인터럽트 처리 루틴
Figure 3. Software interrupt handling routine

2.4 uC/OS-II 초기화

uC/OS-II는 함수 main에서 수행을 시작한다. 함수 main은 함수 OSInit을 불러 커널이 사용하는 모든 자료구조를 초기화한 후, 함수 OSTaskCreate을 불러 여러 태스크를 생성할 태스크인 TaskStart를 생성한다. 이때 생성된 태스크 TaskStart는 준비 상태(ready)에 있으며 아직 수행은 하지 않는다. 그 후 함수 main은 함수 OSStart를 불러서 멀티태스킹을 시작한다. 멀티태스킹이 시작되면 준비 상태(ready)에 있는 태스크 중 가장 우선순위가 높은 태스크인 TaskStart를 처음으로 수행시키며, 이 태스크는 함수 InterruptEnable을 불러 앞에서 설명한 타이머 인터럽트 및 소프트웨어 인터럽트를 활성화시키고, 여러 태스크(Task1, Task2, ...)를 생성하여 커널은 정상 수행 상태가 된다. 그림 4는 함수 main 및 태스크 TaskStart이다.

```
int main(void)
{
  OSInit();                    // uC/OS-II 자료구조 초기화
  OSTaskCreate(TaskStart, ···); // 태스크 TaskStart 생성
  OSStart()                    // 멀티 태스킹 시작
}

void TaskStart(...)            // 태스크 TaskStart
{
  InterruptEnable()            // 인터럽트 활성화
  OSTaskCreate(Task1, ···)     // 태스크 Task1 생성
  OSTaskCreate(Task2, ···)     // 태스크 Task2 생성
  ...
}
```

그림 4. 함수 main 및 태스크 TaskStart
Figure 4. Function main and task TaskStart

III. 하이퍼바이저 설계

하이퍼바이저는 운영체제가 관리하는 하드웨어 자원을 가상화한다[9]. 본 연구에서 다루는 uC/OS-II 커널은 마이크로프로세서와 메모리 자원만을 관리하기 때문에 본 연구에서 개발하는 하이퍼바이저도 마이크로프로세서 및 메모리의 가상화에 중점을 준다. 마이크로프로세서 자원은 2.3에서 기술한 타이머 인터럽트 및 소프트웨어 인터럽트를 제어하여 가상화하고 메모리 자원은 uC/OS-II 커널이 가상 메모리를 사용하지 않고 실제 메모리를 사용하기 때문에 물리적 메모리를 커널 수에 따라 파티션하여 사용함으로써 가상화한다. 본 연구에서 개발하는 하이퍼바이저의 주요 구성 요소는 표 1과 같다.

표 1. 하이퍼바이저의 구성 요소 및 기능 설명
Table 1. Hypervisor components and functions

구성 요소	기능 설명
인터럽트 제어 루틴	각 커널의 타이머 및 소프트웨어 인터럽트 처리 루틴을 제어하는 루틴
시스템 초기화 코드	하이퍼바이저 및 각 커널을 정상 수행 상태까지 유도하는 코드
커널 간 통신 API	각 커널 사이 데이터를 전달하거나 받는 일을 처리하는 API

3.1 인터럽트 제어 루틴

하나의 커널이 수행되는 환경에서는 하나의 인터럽트에 하나의 처리 루틴이 연결되지만 여러 커널이 동시에 수행되는 환경에서는 인터럽트 자원을 가상화하여 여러 커널이 사용하여야 한다. 이 환경에서는 각 커널의 인터럽트 처리 루틴은 직접 인터럽트 자원을 제공받을 수는 없고 하이퍼바이저의 제어에 의하여 인터럽트 자원을 제공받는다. 하이퍼바이저의 인터럽트 제어 루틴은 시스템의 실제(real) 인터럽트 처리 루틴으로 동작하면서 인터럽트가 발생하면 하나의 커널을 선택하여 선택된 커널의 가상(virtual) 인터럽트 처리 루틴에게 인터럽트를 넘기는 역할을 한다. 각 커널의 가상 인터럽트 처리 루틴은 기존 하나의 커널 수행 환경에서 주어진 인터럽트의 인터럽트 처리 루틴을 그대로 사용한다. 그림 5 및 그림 6은 하이퍼바이저의 타이머 및 소프트웨어 인터럽트 제어 루틴들의 흐름도이다.

하이퍼바이저의 타이머 인터럽트 제어 루틴은 타이머 인터럽트 자원을 라운드 로빈 방식을 사용하여 돌아가면서 각 커널에게 제공한다. 만약 n개의 커널이 수행 중인 환경에서 현재 수행 중인 커널의 번호가 k(1<=k<=n)일 때 타이머 인터럽트가 발생하면 하이퍼바이저의 타이머 제어 루틴은 이 인터럽트를 커널 ((k+1) modulo n)에게 전달한다. 타이머 인터럽트는 현재 1/100 초를 주기로 발생하기 때문에 각 커널은 n/100초 주기로 이 인터럽트를 제공받는다. 이 정책은 각 커널의 자원 활용률을 고려하여 수정 가능하다. 본 논문에서는 인터럽트를 각 커널에게 분배하는 정책에 관하여 다루지는 않는다.

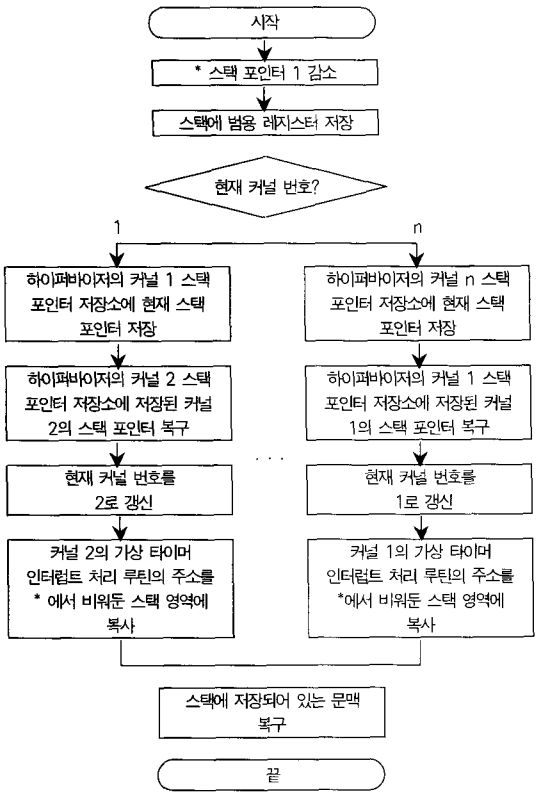

그림 5. 하이퍼바이저의 타이머 인터럽트 제어 루틴의 흐름도
Figure 5. Flow diagram of timer interrupt control routine in hypervisor

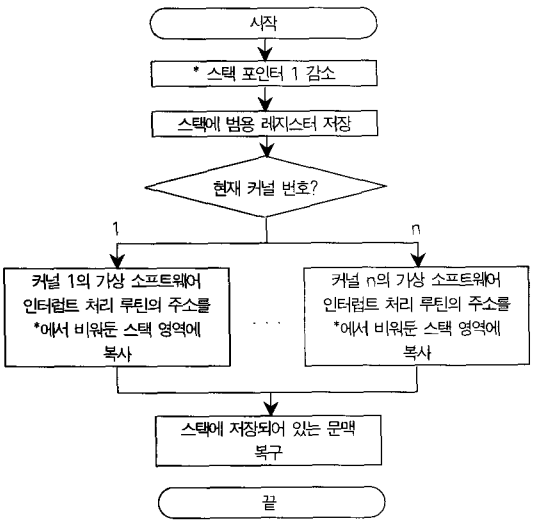

그림 6. 하이퍼바이저의 소프트웨어 인터럽트 제어 루틴의 흐름도
Figure 6. Flow diagram of software interrupt control routine in hypervisor

소프트웨어 인터럽트는 특별한 명령어의 수행(예: SWI 명령어)에 의하여 발생하는 동기적인(synchronous) 인터럽트이기 때문에 앞에서 설명한 타이머 인터럽트와 달리 발생한 인터럽트 자원을 인터럽트를 발생시킨 커널에게 제공한다. 즉 어떤 커널 k(1<=k<=n)를 수행하다가 소프트웨어 인터럽트가 발생할 경우 커널 k의 소프트웨어 인터럽트 서비스 루틴으로 제어를 넘긴다.

3.2 시스템 초기화 코드

하이퍼바이저의 시스템 초기화 코드는 각 커널의 스택 주소 및 각 커널의 함수 main의 주소를 저장하고, 각 커널의 함수 main을 커널 번호 순서대로 불러서 각 커널의 첫 task인 TaskStart의 시작 지점까지만 수행시킨 후 하이퍼바이저로 돌아와서 하이퍼바이저 내에 정의된 함수 InterruptEnable을 불러서 타이머 및 소프트웨어 인터럽트를 설정한 후 커널 1의 첫 태스크인 TaskStart를 부르는 일을 한다. 여기서 중요한 것은 각 커널이 가지고 있는 함수 InterruptEnable은 수행시키지 않고 이에 대응하는 하이퍼바이저의 함수 InterruptEnable을 대표로 수행시킨다는 점이다. 각 인터럽트를 설정하고 활성화하는 일은 하이퍼바이저만이 할 수 있기 때문이다. 그림 7은 하이퍼바이저의 시스템 초기화 과정의 흐름을 그림으로 나타낸 것이다.

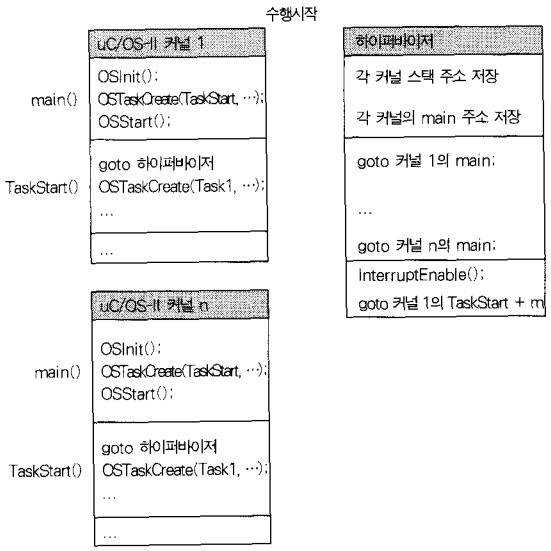

그림 7. 하이퍼바이저의 시스템 초기화 과정 흐름도
Figure 7. Flow diagram of initialization in hypervisor

3.3 커널 간 통신 API

하나의 마이크로프로세서 상에서 하나의 커널이 수행되는 환경과는 달리 하나의 마이크로프로세서 상에서 여러 개의 커널이 동시에 수행되는 환경에서는 커널 간 데이터 전달을 위한 API가 필요하다. 본 연구에서 개발한 하이퍼바이저는 표 2와 같은 커널 간 데이터 전달을 위한 API를 제공한다. 본 API는 실험을 위하여 개발한 간단한 API이지만 사용자의 요구에 따라 확장하여 구현할 수 있다.

구성 요소	기능 설명
OSKernelWrite (receiver, data, length)	현재 커널이 receiver 커널에게 data를 보낸다.
OSKernelRead (sender, data, length)	현재 커널이 sender 커널이 보낸 data를 받는다.

표 2. 커널 간 통신 API
Table 2. API for inter-kernel communication

다음 그림 8은 이들 API를 구현한 코드이다. 커널 간 전달하는 데이터의 저장을 위한 메모리는 실시간 특성의 제공을 위하여 하이퍼바이저의 데이터 섹션에 정적 크기로 할당된 메모리 풀을 사용한다.

```
U16 OSKernelWrite(INT8U receiver, void *data, U32 length)
{
  인터럽트를 비활성화 시킨다.
  커널 receiver의 큐에 data 및 length를 추가한다
  인터럽트를 활성화 시킨다.
  성공 혹은 오류 코드를 return한다.
}

U16 OSKernelRead(INT8U *sender, void *data, U32 *length)
{
  인터럽트를 비활성화 시킨다.
  성공 혹은 오류 코드를 return한다.
}
```

그림 8. 커널 간 통신 API 루틴
Figure 8. API Routines for inter-kernel communication

3.4 가상화 환경에서 메모리 모습

2개 이상의 커널이 동시에 수행되는 가상화된 수행 환경에서는 각 커널은 자신의 (가상) 인터럽트 처리 루틴, 수행을 위한 코드 및 데이터 그리고 스택을 가지고 있다. 또한 하이퍼바이저는 가상 수행 환경을 구성하기 위하여 인터럽트 제어 루틴, 하이퍼바이저를 포함한 각 커널의 현재 스택 포인터 값을 저장하는 장소 및 커널 사이의 데이터 통신을 위하여 데이터를 저장하는 큐 그리고 스택을 가지고 있다. 그림 9는 2개 이상의 커널이 수행되는 가상 환경에서의 메모리 모습을 나타낸 그림이다.

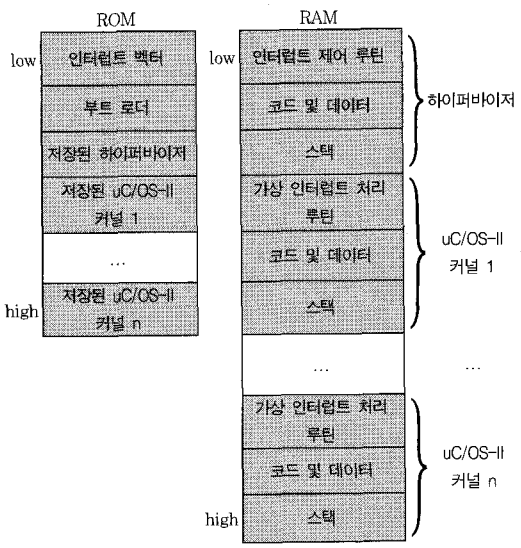

그림 9. n개의 uC/OS-II 실시간 커널이 수행되는 메모리
Figure 9. Memory for n uC/OS-II kernels

IV. uC/OS-II 포팅

하나의 마이크로프로세서 상에서 수행되던 uC/OS-II 실시간 커널은 하이퍼바이저 상에서 수행되기 위하여 조금 수정되어야 한다. 이 수정은 앞의 3.2에서 설명한 시스템 초기화 코드의 흐름에 따라 각 커널이 시작되어 수행될 수 있도록 하기 위함이다. 각 커널의 수정할 부분을 요약하면 다음과 같다.

각 커널에서 사용되는 매크로 OS_TICKS_PER_SEC 값을 기존의 값을 커널의 수로 나눈 값으로 재조정한다. 이는 타이머 인터럽트가 발생하더라도 각 커널이 돌아가면서 이 인터럽트의 서비스를 받기 때문이다.

각 커널에서 첫 태스크인 TaskStart에서 인터럽트를 활성화하기 위하여 함수 InterruptEnable을 부른다(그림 4 참고). 이 부분은 하이퍼바이저가 불러서 활성화하기 때문에 제거한다(그림 7 참고).

각 커널의 첫 태스크인 TaskStart는 각 커널의 초기화가 끝나면 제어를 하이퍼바이저로 돌려야 한다. 이를 위하여 하이퍼바이저로 제어를 넘기는 코드의 삽입이 필요하다(그림 7 참고).

V. 구현 및 시험

본 논문에서는 3장 및 4장에서 설계한 내용을 Jupiter 32비트 EISC(Extensible Instruction Set Computer) 마이크로프로세서가 탑재된 "Jupiter EVM 보드" 상에서 구현하였다. 이 보드는 저장장치로 EPROM 1M바이트, NAND Flash 8M바이트, SDRAM 8M바이트가 장착되어 있으며, LED, LCD, 7 세그먼트, UART, USB, 이더넷 등의 장치가 장착되어 있다. 개발된 하이퍼바이저는 C 프로그램 약 350줄 어셈블리 프로그램 약 400줄로 구성되어 있다. 본 구현에서는 마이크로프로세서의 속도(50MHz) 및 주 메모리 용량(8M바이트)을 고려하여 동시에 수행되는 uC/OS-II 실시간 커널의 수를 3으로 설정하였다. 그림 10은 Jupiter 32비트 EISC 개발 보드의 사진이다.

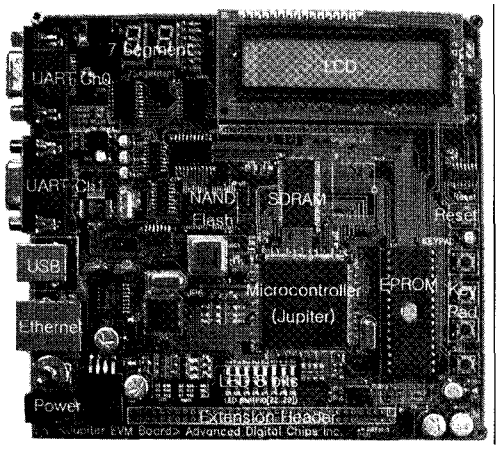

그림 10. Jupiter EVM 보드
Figure 10. Jupiter EVM Board

5.1 실시간 동작 시험

uC/OS-II 실시간 커널의 가장 중요한 기능은 실시간 제어 기능이다. 본 시험은 하이퍼바이저 상에서 가상화되어 수행되는 각 커널의 실시간 기능이 정확하게 동작되는 지를 시험하는 것이다. 본 시험을 위하여 하이퍼바이저 상에 3개의 커널을 동시에 수행시켰으며, 각 커널에는 10개의 태스크를 생성하고, 각 태스크는 자연수 1과 20 사이의 난수(random number) n을 생성하고, 커널 함수 OSTimeDly를 사용하여 n tick 시간만큼의 지체(delay)를 가진 후, 고정 양의 CPU 연산을 수행하는 함수 cpuwork를 부르게 프로그램하였다. (주: 일반적인 실시간 태스크는 모두 이와 유사한 실행을 수행함.) 이 실험에서 함수 cpuwork()는 각 커널의 CPU 활용도가 25% 정도로 유지되게 프로그램하였다. 그림 11은 시험에 사용한 태스크이다. 이 시험 프로그램을 10시간 이상 수행시킨 결과 함수 OSTimeDly에 주어진 tick 값인 n과 커널 함수 OSTimeDly를 부르기 전과 부르고 난 뒤 tick 값의 변화 값이 항상 같아서 함수 printerror가 불리지 않고 정확하게 시간이 지체됨을 확인할 수 있었다.

5.2 독립 수행 환경 시험

가상화의 장점 중 하나는 가상화되어 수행되는 각 커널이 서로 독립된 수행 환경을 가진다는 점이다. 예를 들어 하나의 커널에서 CPU 활용도가 높아지더라도 다른 커널에 영향을 주지 않아야 한다. 본 시험을 위하여 하나의 하이퍼바이저 상에 3개의 커널을 동시에 수행시켰으며, 각 커널에는 10개의 사용자 태스크를 생성하여 수행시켰다. 각 태스크는 그림 11과 유사한 태스크지만 본 시험을 위하여 함수 cpuwork를 수정하여 각 커널의 CPU 활용도가 각각 10%, 40%, 80% 정도로 차이가 나게 프로그램을 작성하였다. 이 시험 결과 각 커널의 CPU 활용도가 초기에 설정된 값을 계속 유지하면서 다른 커널에 간섭을 주지 않고 독립적으로 수행됨을 보였다.

```
void Task (...)
{
  while(1) {
    n = random(20);          // 난수 생성하여 n에 할당
    n1 = OSTimeGet()         // 현재 tick 값을 가져옴
    OSTimeDly(n);            // n tick 시간만큼 태스크 지체
    n2 = OSTimeGet()         // 현재 tick 값을 가져옴
    if(n != (n2-n1)) printerror()  //실시간성에이상있으면출력
      cpuwork();             // CPU 연산 수행
  }
}
```

그림 11. 실시간 시험을 위한 태스크
Figure 11. Task for real-time test

5.3 커널 간 통신 시험

개발한 하이퍼바이저는 커널간 데이터 통신을 위한 API가 포함되어 있다. 본 시험을 위하여 하나의 하이퍼바이저 상에 3개의 커널을 동시에 수행시켰으며, 각 커널에는 10개의 사용자 태스크를 생성하여 수행시켰다. 각 태스크는 그림 11과 유사한 태스크지만 본 시험을 위하여 함수 cpuwork을 수정하여 각 커널의 CPU 활용도가 각각 평균 10%, 40%, 80% 정로로 차이가 나게 프로그램을 작성하

였다. 커널 2 및 커널 3는 초당 1회 자기 커널의 CPU 활용도 수치를 커널 1에게 전달하게 프로그램하였다. 커널 1은 커널 2 및 커널 3이 전달한 CPU 활용도를 매초 읽어서 UART0으로 출력하게 프로그램하였다. 본 시험 결과 커널 간 데이터 통신이 설계된 바와 같이 정확하게 수행됨을 볼 수 있었다. 그림 12는 이 시험 시 커널 1이 UART0을 통하여 출력하는 데이터를 보여주는 화면이다. (주: 이 그림에서 OS1, OS2 및 OS3은 각각 커널 1, 커널 2 및 커널 3을 의미한다.)

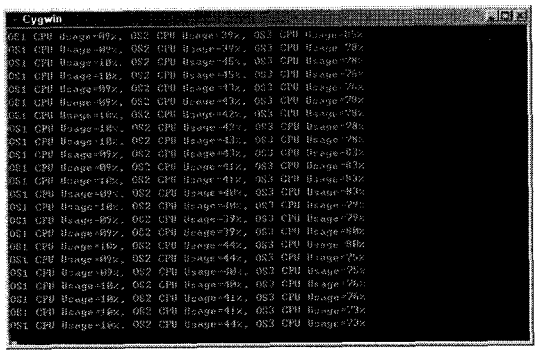

그림 12. 커널 간 통신 시험 화면 (커널 1의 출력 화면)
Figure 12. Test screen of inter-kernel communication

5.4 전력 소비량 시험

현재 대부분의 소형 내장형 마이크로프로세서가 저전력을 위한 기능(예: sleep 모드 지원 등)을 가지고 있지 않는 현실에서 여러 마이크로프로세서를 사용하여야 하는 내장형 응용 분야의 경우 가상화를 사용하여 사용 마이크로프로세서의 수를 줄이면 이론적으로 전체 시스템의 전력 소비량을 줄일 수 있다. 본 절에서는 이를 가상화를 통한 실제 측정으로 확인해 보았다. 본 시험은 하나의 개발 보드 상에서 3개의 커널을 수행시키는 경우와 3개의 개발 보드에서 각각 하나의 커널을 수행시킬 경우 소비되는 전력량을 측정하여 비교하였다. 사용된 프로그램은 앞의 5.2와 같이 하나의 커널에 10개의 태스크를 생성하고 각 커널의 CPU 활용도가 각각 10%, 40%, 80% 정도가 유지되게 프로그램을 설정하여 수행시켰다. 측정 방법은 각 보드의 전원부(직류 5V 사용)에 전류 측정기를 부착하여 시스템이 수행되는 도중 소비되는 전류 값을 측정하였다. (주: 소비 전력량은 소비 전류 값에 비례함.) 본 시험에서는 개발 보드에 장착된 장치인 LED, LCD, 7 세그먼트, UART, USB, 이더넷을 비활성화 하여 사용되지 않는 상태에서 측정되었다. 시험 결과 하나의 보드 상에서 3개의 커널을 동시에 수행시키는 경우 평균 39.3667mA의 전류가 측정되었고, 3개의 보드에 각각 하나의 커널을 수행하는 경우 각 보드 당 평균 39.6706mA, 39.6874mA 및 39.6997mA의 전류가 측정되었다. 시험의 객관성을 보장하기 위하여 측정은 보드 당 0.5초 마다 한 번씩 측정하고 각 보드 당 측정 회수는 200회로 설정하였다. 측정 결과 하나의 보드에 3개의 커널을 동시에 수행시키는 경우와 하나의 보드에 하나의 커널을 수행시키는 경우 소비 전류는 거의 같음을 알 수 있다. (주: 현재 uC/OS-II 실시간 커널은 각 커널의 CPU 활용도가 떨어지더라도 CPU를 저전력 모드로 유도하는 기능이 없음.) 그림 12는 3개의 커널을 동시에 수행시키는 시스템과 하나의 커널을 수행하는 시스템이 각각 소비하는 전류 값을 측정한 결과를 그래프로 나타낸 그림이다.

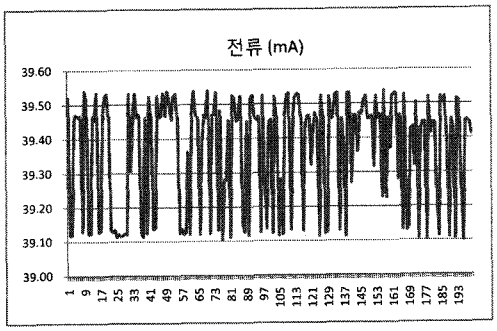

그림 12-A. 3개의 커널 수행 시 전류 측정 값
Figure 12-A. Currents measured on three kernels

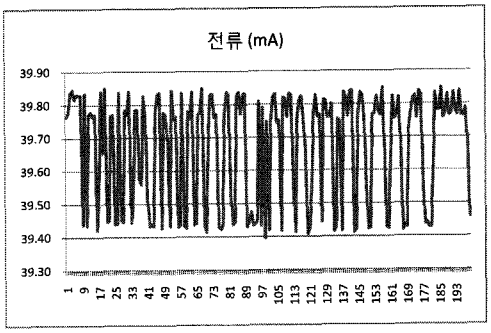

그림 12-B. 하나의 커널 수행 시 전류 측정 값
Figure 12-B. Currents measured on one kernel

5.5 시스템 부피 및 무게 감소 효과

본 연구에서 개발한 하이퍼바이저를 많은 실시간 내장형 시스템이 결합되어 사용되는 응용 분야에 적용할 경우 시스템의 부피 및 무게의 감소 효과를 얻을 수 있다. 예를 들어 최근 개발되는 고가의 차량의 경우에 수십 개의 내장형 마

이크로프로세서가 사용되고 있으며 이 중 많은 마이크로프로세서가 실시간 커널을 사용하고 있다. 이러한 차량 시스템에 하이퍼바이저가 사용되면 차량 내 내장형 시스템이 장착되는 부분의 부피 및 무게를 감소시켜서 차량의 공간 활용도가 높아질 수 있다.

VI. 결론

본 논문에서는 내장형 시스템에 사용되는uC/OS-II 실시간 커널을 가상화하는 하이퍼바이저를 개발하였다. 개발된 하이퍼바이저는 하나의 마이크로프로세서를 다수의 커널이 동시에 사용할 수 있게 하기 위하여 각 커널의 타이머 인터럽트 처리 루틴 및 소프트웨어 인터럽트 처리 루틴을 제어하는 알고리즘을 통하여 마이크로프로세서를 가상화하였고 실제 메모리는 파티션하는 방식을 사용하여 가상화하였다. 본 논문에서 설계한 하이퍼바이저를 Jupiter 32비트 EISC 마이크로프로세서가 탑재된 "Jupiter EVM 보드" 상에서 구현하여 실시간 동작 시험, 독립 수행 시험, 커널 간 통신 시험 및 전력 소비 시험을 하였고, 시스템 부피 및 무게 감소 효과에 대하여도 기술하였다. 시험 결과 가상화된 커널이 정상적으로 수행됨을 확인하였고 하나의 하이퍼바이저에 다수의 커널을 동시에 수행시킬 경우 전력 소비량을 줄일 수 있음을 확인하였다. 본 연구 결과는 다음과 같은 장점을 제공한다. 하나의 하드웨어 시스템에 여러 개의 커널을 동시에 수행시키므로 하드웨어 가격 절감 효과를 얻을 수 있다. 또한 내장형 시스템이 차지하는 부피 및 무게를 줄여서 많은 내장형 시스템이 탑재되는 응용 분야에 활용할 경우 공간 활용 면에서 매우 유리하다. 또 본 논문에서도 시험한 바와 같이 전력 절감 효과를 얻을 수 있다. 본 연구팀은 이번 연구를 바탕으로 서로 다른 기종의 내장형 운영체제(예: uC/OS-II, uCLinux[16], 실시간 Linux[17] 등)를 동시에 수행시키는 하이퍼바이저를 현재 연구하고 있으며, 서로 다른 기종의 내장형 운영체제가 동시에 수행되는 하이퍼바이저가 개발되면 활용 범위는 더 커질 것으로 전망된다.

참고문헌

[1] A. Singh, An Introduction to Virtualization, KernelThread.com, 2004. 2.

[2] J. E. Smith and R. Nair, Virtual Machines, 2005.

[3] F. J. Corbató, M. Merwin-Daggett, and R. C. Daley, An Experimental Time-sharing System, Proceedings of Spring Joint Computer Conference, pp. 335 - 44, 1962. 5.

[4] P. J. Denning, Virtual Memory, ACM Computing Surveys, Volume 2, Issue 3, pp. 153-189 , ACM Press, 1970. 9.

[5] D. A. Patterson, G. Gibson and R. H. Katz, A Case for Redundant Arrays of Inexpensive Disks (RAID), Proceedings of the ACM SIGMOD, pp. 109-116, 1988.

[6] A. Chernoff, M. Herdeg, R. Hookway, C. Reeve, N. Rubin, T. Tye, S. B. Yadavalli and J. Yates, FX!32 A Profile-Directed Binary Translator, IEEE Micro, Volume 18, Number 2, pp. 56-64, 1998.

[7] S. Herrod, The Future of Virtualization Technology, The 33rd Annual International Symposium on Computer Architecture, 2006.

[8] S. Crosby and D. Brown, The Virtualization Reality, Queue, Volume 4, Issue 10, pp. 34-41, ACM Press, 2006.

[9] M. Rosenblum and T. Garfinkel, Virtual Machine Monitors: Current Technology and Future Trends, Computer, Volume 38, Number 5, pp. 39-47, IEEE Press, 2005.

[10] J. J. Labrosse, MicroC/OS-II The Real-Time Kernel Second Edition, CMP Books, 2002.

[11] Micro C/OS-II Real-Time Kernel, http://www.micrium.com/products/rtos/kernel/rtos.html, 2007.

[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, Xen and the Art of Virtualization, Proceedings of the ACM Symposium on Operating Systems Principles, pp. 164-177, 2003.

[13] VMware Workstation, http://www.vmware.com/products/desktop/workstation.html, VMware, Inc., 2007.

[14] Advanced Digital Chips Inc., Jupiter Information Internet Appliance 32-bit EISC Microcontroller, 2004. 9.

[15] Advanced Digital Chips Inc., Jupiter Education Board Manual, Version 1.0, 2004. 11.
[16] Embedded Linux/Microcontroller Project, http://www.uclinux.org, 2007.
[17] 김성락, 실시간 Linux 환경에서 효율적인 스케줄링을 위한 선택 알고리즘의 구현, 컴퓨터정보학회 논문지, 7권 2호, 2002.

저 자 소 개

신동하

1980년 경북대학교 전자공학과 학사
1982년 서울대학교 전자계산기공학과 석사
1994년 University of South Carolina 컴퓨터과학과 박사
1982년 ~ 1996년 한국전자통신연구원 연구원
1997년 ~ 현재 상명대학교 소프트웨어학부 부교수
관심 분야: 가상화 기술, 임베디드 컴퓨팅, 논리 및 함수 프로그래밍, 리눅스 및 윈도우즈 시스템 프로그래밍

김지연

2006년 상명대학교 소프트웨어학부 학사
2006년 ~ 현재 상명대학교 일반 대학원 컴퓨터과학과 석사 과정
2007년 ~ 현재 (주) 마크애니 연구원
관심 분야: 가상화 기술, 임베디드 컴퓨팅, 윈도우즈 및 리눅스 시스템 프로그래밍