실시간 운영체제에서 효율적인 디버그 정보 관리를 위한 버퍼 설계 및 구현

이재규°, 류현수, 정명조, 성영락*, 이철훈
충남대학교 컴퓨터공학과, *국민대학교 전자정보통신공학부
{jklee°,hsryu,mjjung,chlee}@ce.cnu.ac.kr, *yeong@mail.kookmin.ac.kr

Design and Implementation of Buffers for Efficient Management of Debug Information in Real-Time Operating Systems

Jae-Gyu Lee°, Hyeon-Soo Ryu, Myoung-Jo Jung, Yeong-Rak Seong* and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
*School of Electrical Engineering, Kookmin Univ.

요 약

실시간 운영 체제(Real-Time Operating Systems)는 시스템 동작이 논리적 정확성뿐만 아니라 시간적 정확성에도 좌우되는 운영 체제이다. 그리고, 실시간 운영체제는 멀티태스킹(Multitasking) 과 ITC(Inter Task Communication)을 제공한다는 점에서 일반 운영 체제인 Windows, Linux, Unix등과 같지만 시간 결정성을 보장해야 한다는 점에서 일반 운영 체제와 다르다. 이러한 실시간 운영체제에서 프로그래머가 디버그 정보를 알기 위해서 여러 가지 기법을 사용하게 된다. 본 논문은 실시간 운영체제에서 시간 결정성을 지키면서 메모리에 관련된 디버그 이벤트들을 알기위한 버퍼의 설계 및 구현에 대해 기술한다.

1. 서 론

실시간 운영 체제는 내장형 시스템에 사용되는 대표적인 운영 체제이다. 내장형 시스템은 프로세서의 처리 속도, 메모리, 전원 등의 자원이 제한된 환경에서 미리 정해진 특정한 기능을 수행하도록 설계된 시스템이다. 다시 말해 특정한 기기에 주어진 작업을 수행하도록 구동 시키는 시스템이라고 할 수 있다. 따라서 이러한 내장형 시스템을 설계하거나 개발할 때 고려 사항들로는 첫째, 최악의 상황에서도 정해진 시간내에 동작해야 하고 둘째, 작은 기기의 용량에 맞도록 설계되어야 하므로 가볍고 효율적이어야 하며 셋째, 위험성을 내포하고 있는 불안정한 상황에서도 오류없이 안정적으로 동작되도록 해야 하며 넷째, 저가의 비용으로 높은 효율을 발휘하도록 해야 한다는 점 등이 있다. 실시간 운영 체제의 이러한 특징 때문에 설계자는 발생 가능한 여러 가지 시나리오를 모두 인지한 상태에서 시스템 개발을 해야 한다.

이렇게 개발된 실시간 운영체제를 이용하여 프로그램을 개발할 때에는 일반적인 응용 프로그램을 개발하는 것보다 더 어려운 디버깅 과정을 겪게 된다. 초기에는 소스코드 디버거를 사용하여 코드 단계에서의 오류를 제거할 수 있다. 하지만 코드 단계의 오류를 제거하고 난 후에 발행 할 수 있[1]는 논리적인 오류에서는 소스코드 디버거를 사용하는 것이 비효율적 이다. 논리적인 오류는 프로그램을 시작하고 프로그램이 얼마동안 동작하면서 언제 오류가 발생할지 알 수 없기 때문이다. 프로그래머가 이러한 논리적인 오류를 찾고 수정하기 위해서는 오류 발생 직전에 소프트웨어에서 어떤 일이 발생했는지 알아야할 필요가 있다. 따라서 본 논문은 실시간 운영체제에서 이러한 디버그 정보를 효율적으로 관리하기 위한 버퍼를 어떻게 설계하고 구현하였는지를 기술하고 있다. 2장에서는 관련연구를, 3장에서는 디버그 정보 관리 버퍼의 설계 및 구현을, 4장에서는 실시간 운영체제 iRTOS에서의 테스트 및 결과를, 5장에서는 결론 및 향후 연구 과제에 대하여 논하였다.

2. 관련연구

2.1 실시간 운영체제 개요

실시간 운영 체제는 시간 결정성과 작은 크기의 실행 이미지를 특징으로 한다. 멀티태스킹(Multitasking)을 지원하고 태스크에 우선순위를 할당하여 높은 우선순위의 태스크가 CPU를 선점하는 스케줄링 방식을 사용한다. 그리고, ITC(Inter Task Communication) 환경은 태스크간 동기화 및 통신을 위하여 세마포어(Semaphore), 메시지 메일박스(Message Mailbox), 메시지 큐(Message Queue) 등을 지원한다.

1) 본 논문은 산업자원부 중기거점과제 연구비 지원에 의한 것임

2.2 iRTOS

iRTOS는 (주)아이네스테크에서 개발한 실시간 운영체제로써 다음과 같은 특징이 있다.

① 멀티태스킹 지원
② 우선순위 기반의 선점형 스케줄링 사용
③ ITC를 위한 세마포어, 메시지 메일박스, 메시지 큐 지원
④ 동적 할당을 위한 힙과 메모리 풀 지원
⑤ Small Foot Print : 약 23KB

3. 디버그 정보관리 버퍼의 설계 및 구현

3.1 printf() 함수를 사용한 디버그 정보 출력시 문제점

많은 프로그래머들이 오류조건 발생 전의 소프트웨어 이벤트들을 알기 위해서 printf()함수를 사용한다. 하지만 printf()함수를 사용한 디버깅 기술은 많은 CPU사이클을 소모한다. 이것은 실시간 운영체제에서 시간 결정성을 보장하지 못하고, 또 다른 오류를 발생 시킬 수도 있다. printf()함수를 사용한 디버깅은 다음과 같은 문제점을 가지고 있다.

①ISR루틴에서 printf()함수의 호출은 불가능하다.

②printf()는 시리얼 포트를 정체 시킬 수 있어서 다른 입출력의 무효화 가능성이 있다.

③시스템 초기화 또는 부팅시의 디버그 정보를 보기위한 printf()함수는 사용 불가능 하다.

3.2 디버그 정보 구조체

printf()함수를 사용한 디버그 정보 출력의 대안으로 디버그 정보관리 버퍼를 사용해서 오류발생전의 소프트웨어 이벤트를 알 수 있게 하였다. 이 기법은 시스템의 타이밍에 최소한의 영향을 미치며, 계산가능한 양의 메모리 자원만을 필요로 한다.

디버그 정보관리 버퍼를 사용한 디버깅 기술은 고정된 크기의 메모리를 사용한다. 각 디버그 정보를 위한 구조체는 다음과 같이 설계 하였다 [그림 1].

```
typedef struct {
        unsigned int time;
        char *event;
        unsigned int data;
} DebugItem;
```

[그림 1] 디버그 정보 구조체

time 필드는 특정 시간의 이벤트와 연관된다. 모든 시스템은 타이머 인터럽트를 주기적으로 발생시키게 되는데 이벤트가 발생하면 타이머의 값을 읽어서 DebugItem.time필드에 기록 한다. time 필드의 값을 확인함으로써 특정 ISR이나 함수를 수행하는데 얼마나 시간이 걸렸는지 확인 알 수 있다. event 필드는 위치에 관한 정보를 제공해주는 문자열 포인터 이다. 이벤트가 발생하면 문자열을 지시하는 포인터만 전달된다. 나중에 디버그 정보관리 버퍼를 출력하면 event포인터를 참조해서 완결된 문자열을 출력한다. data 필드는 이벤트와 함께 지역 또는 전역변수와 연관해서 사용된다. 예를 들면 ISR은 인터럽트 이벤트 레지스터의 값을 포함하는 지역변수를 사용한다. 다른 예로 현재 시스템의 정보를 유지하기 위한 전역변수의 사용이다. 이벤트가 발생하면 data 필드의 값을 버퍼의 메모리로 복사한다.

3.3 디버그 정보관리 버퍼

디버그 정보관리 버퍼는 DebugItem의 배열로 구현하였다. 두개의 정수형 전역 변수를 사용해서 버퍼의 시작과 다음 이벤트가 발생했을 때 사용할 슬롯을 추적한다. 버퍼의 사용과 관련해서 다음과 같은 함수를 구현 하였다.

①buf_init() : 정보관리 버퍼를 초기화 [그림 2]

```
void buf_init() {
        bufstart = 0;
        bufnext = 0;
        buf_add("init");
}
```

[그림 2] buf_init()

②buf_add(char *event, int data) : 이벤트 발생시에 정보들을 버퍼에 저장 [그림 3]

```
void buf_add(chat *event, int data) {
        MK_InterruptDisable();
        buf[bufnext].time=MK_getTick();
        buf[bufnext].event=event;
        buf[bufnext].data=data;
        if(++bufnext >= BUFSIZE) {
        bufnext=0;
        if(bufnext==bufstart) {
          if(++bufstart >= BUFSIZE)
            bufstart=0; }
        }
       MK_InterruptRestore();
}
```

[그림 3] buf_add()

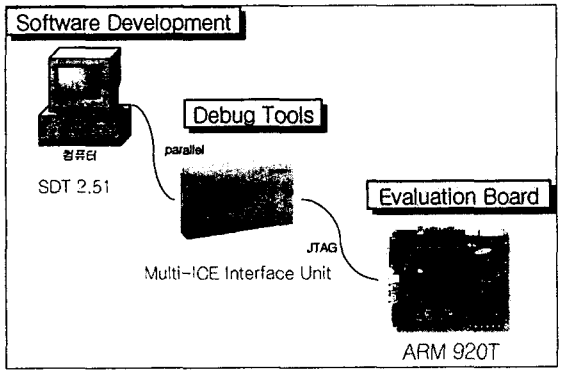
③buf_print() : 버퍼의 모든 정보를 출력 [그림 4]

```
void hist_print()
{
   int start = bufstart;
   while ( start != bufnext )
   {
MK_printf("%d:%s:%d",buf[start].time,buf[sta
rt].event,buf[start].data);
   MK_printf()("₩n₩r");
   start++;
   if ( start >= BUFSIZE ) {
     start = 0;}
   }
}
```

[그림 4] buf_print()

4. iRTOS에서의 테스트 및 결과

본 논문에서 구현한 디버그 정보관리 버퍼의 소스파일인 Debug.h와 Debug.c를 iRTOS의 커널에 포함시켜 컴파일 하고 테스트 프로그램을 작성하여 테스트 하였다. 테스트는 ARM 920T 기반의 S3C2400 Evaluation Board로 진행 하였다 [그림 5].



[그림 5] 테스트 환경

테스트 프로그램을 작성하고 buf_add()함수를 사용하여 몇 개의 이벤트를 등록한 후 buf_print()함수를 통하여 다음과 같은 이벤트의 결과를 볼 수 있었다 [그림 6]

```
2010:read
2012:read,wating for data
2320:isr,RxReq
....
```

[그림 6] 테스트 결과

5. 결론 및 향후 연구 과제

본 논문에서 구현한 디버그 정보관리 버퍼를 사용해서 코드단계의 오류를 수정한 후의 논리적인 오류를 실시간 운영체제의 여러 가지 요구사항을 위배하지 않고 확인함으로써 좀더 효율적인 디버깅 환경을 구축할 수 있었다. 향후에는 현재 사용자 프로그램의 메모리 사용 현황까지 확인 가능한 실시간 디버그 정보관리 버퍼를 연구할 계획이다.

참고문헌

[1] http://www.inestech.com

[2] D. Comer, *Operation System Design VOL 1 : THE XINU APPROACH*, PRENTICE HALL, 1988.

[3] DAN HILDEBRAND, "Memory Protection in Embedded Systems, Embedded Systems Programming, 1996.

[4] ANDY PURCELL, "Debug Tip", Embedded Systems Programming, 2001.

[5] 윤기현, 김용희, 박희상, 이철훈, "Safety-Critical Real-Time Operating System의 설계 및 구현", iRTOS™, 정보과학회 춘계학술발표논문집, 제 30권 1호, 2003.