실시간 운영체제 µRTOS™ 와 Nucleus-Wrapper 의 성능 비교 및 분석

이숭열°, 최인범, 정명조, 이철훈
충남대학교 컴퓨터공학과
{sy-lee°, ibchoi, mjjung, clee}@cnu.ac.kr

Comparison and analysis of Performanc for µRTOS™ and Nucleus-Wrapper

Soong-Yeol Lee°, In-Bum Choi, Myoung-Jo Jung, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University*

요 약

임베디드 시스템 특히 실시간 시스템에서 응용프로그램들은 이식성이 굉장히 낮다. 따라서 실시간 운영체제와 다른 실시간 운영체제 기반의 응용프로그램들 사이에는 이식성이 문제가 된다. 이는 응용프로그램과 실시간 운영체제의 활용과 개발의 한계를 가져오며 이러한 이식성의 문제는 미들웨어의 구현으로 해결 할 수 있다. 하지만 미들웨어의 구현은 Time-latency 와 같은 오버헤드를 발생시킬 수 있다. 본 논문에서는 미들웨어의 추가에 따른 Time-latency 를 측정하고 비교함으로써 이러한 오버헤드와 이식성 사이의 trade-off 를 알아보았다.

1. 서 론

최근 임베디드 시스템 분야에서 실시간 운영체제를 탑재하여 개발된 제품들이 점차 증가하고 있다. 이런 추세로 현재 많은 실시간 운영체제가 등장하고 있으며 각각의 운영체제에 적합한 응용프로그램들이 개발되고 있다. 하지만 이런 상황에서 실시간 운영체제와 응용프로그램과의 호환여부가 큰 문제로 부각되고 있다. 즉, 실시간 운영체제와 다른 실시간 운영체제 기반의 응용프로그램간의 이식성의 문제로 응용프로그램의 개발과 실시간 운영체제의 활용 범위가 제한적일 수 밖에 없다. 따라서 위와 같은 응용프로그램의 시스템 종속성 해소 이외에 소프트웨어 개발 주기의 단축, 소비자 욕구의 증대로 인한 다양한 서비스 제공 기대, 응용프로그램의 수명 장기화 필요성, 소프트웨어의 개발, 유지, 보수의 체계화 필요성 등에 의해 미들웨어의 연구가 필요하다. 하지만 운영체제와 응용프로그램 사이에서 이질적인 분산환경을 추상화시켜 서로 다른 기능을 정합시키는 소프트웨어인 미들웨어의 사용은 Time-latency 와 같은 오버헤드를 야기시킬 가능성이 있다.

본 논문에서는 미들웨어의 사용으로 인한 오버헤드를 측정하기 위하여 실시간 운영체제 중에서 안정성 및 성능을 인정받은 µRTOS™ 와 Nucleus-Wrapper 를 테스트 대상으로 하였으며 태스크 생성/삭제, 메시지 큐의 생성/삭제, 세마포어의 생성/삭제에 대한 시간지연(Time-latency)값을 비교 및 분석하였다.

본 논문의 구성은 2 장에서 관련연구로서 실시간 운영체제 µRTOS™ 와 미들웨어의 한 종류인 Wrapper 의 개념에 대해 알아보고, 3 장에서는 S3C2440 의 Timer 를 이용하여 µRTOS™ 과 Nucleus-Wrapper 의 성능을 비교하는 방법을 제시하였다. 4 장에서는 테스트 환경 및 결과를, 5 장에서는 결론 및 향후 연구 과제를 기술한다. [6]

2. 관련 연구

2.1 µRTOS™ 커널

2.1.1 멀티태스킹(Multitasking)

멀티태스킹은 여러 개의 태스크가 동시에 수행될 수 있도록 하는 방식이다. 각 태스크는 자신의 문맥(Context)을 가지고 있으며 태크스 제어 블록(TCB, Task Control Block)에 저장된다. [2][4]

2.1.2 태스크 스케줄링(Task Scheduling)

일반적인 실시간 운영체제에서는 태스크가 CPU 를 할당 받는 시점에 따라서 선점형 커널과 비선점형 커널로 구분된다. µRTOS™ 커널은 우선순위 기반의 선점형 스케줄링을 지원하며 동일한 우선순위의 태스크에 대해서는 라운드 로빈 스케줄링과 FIFO 스케줄링 방법을 지원한다. [2][4]

2.1.3 태스크간 통신(Inter-Task Communication)

독립적인 태스크들 사이에서 공유자원 사용에 대한 동기화와 데이터를 교환하기 위해 태스크들 간의 상호 협조할 수 있는 통신 메커니즘이 필요하다. [2][4]

- 전역변수를 사용하는 공유 메모리(Shared Memory)
- 공유자원 접근에 대한 상호배제 및 동기화를 제공

* 본 논문은 국방과학연구소의 실시간 운영체제 인터페이스용 미들웨어 연구과제의 수행결과임.

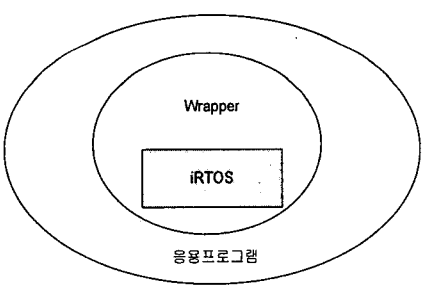
하는 세마포어(Semaphore)
- 태스크들 사이의 메시지를 전달하는 메시지 큐 (Message Queue)
- 예외처리를 위해 비동기적으로 사용하는 시그널 (Signal)

2.1.4 상호배제(Mutual Exclusion)

상호 배제는 공유 데이터에 대해 하나의 태스크가 접근하고 있을 때, 다른 태스크가 접근하지 못하도록 하는 방법을 말하며 아래의 3가지 방법이 있다.
- Interrupt Disable
- Context Switch Disable
- Locking

2.2 래퍼(Wrapper)

응용프로그램 개발자 입장에서 기반 실시간 운영체제가 변경될 경우 응용 프로그램에서 사용하던 이전 실시간 운영체제의 인터페이스 대신 새로운 실시간 운영체제의 인터페이스로 변경해야 하는 번거로움이 발생한다. 따라서 응용프로그램 개발의 연속성, 효율성, 그리고 편리한 유지보수 등을 위해서 응용 프로그램 자체는 변경되지 않는 인터페이스(Wrapper) 제공이 필요하다. [그림 2.1]은 래퍼(Wrapper) 사용을 도식화한 것이다. [5][6]



[그림 2.1] 래퍼(Wrapper)

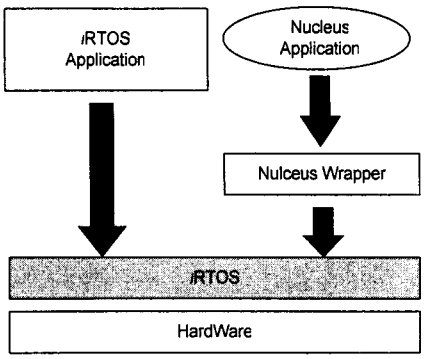
2.3 S3C2440A 의 타이머

S3C2440A 는 ARM920T CPU core 기반의 보드로서 5개의 16 비트 타이머를 가지고 있다. 이와 같은 타이머는 PWM(Pulse Width Modulation) timer 로 각 타이머마다 count buffer register 와 compare buffer register 를 가지고 있어 카운트 초기값을 설정할 수 있으며 이 설정에 의해 타이머 값을 얻을 수 있다.[1]

3. Time-latency 측정 방법의 구현

3.1 설 계

본 논문은 미들웨어의 한 종류인 Necleus-Wrapper 를 사용하여 [그림 3.1]과 같은 환경에서 태스크, 메시지 큐, 세마포어의 생성/삭제 시의 Time-latency 값을 측정하였고 그 값을 비교함으로써 오버헤드를 측정하는 것에 중점을 두어 설계하였다. [그림 3.1]은 Wrapper 가 사용되는 것을 도식화하여 보여주고 있다.



[그림 3.1] Wrapper의 사용

본 논문은 ARM920T CPU 기반의 S3C2440A 보드의 타이머를 사용하여 Time-latency 값을 측정하였고 iRTOS™ 경우 MK_GetSystemClock() 함수를, Nucleus-Wrapper 의 경우 get_usec() 함수를 사용하여 시간지연(Time-latency)값을 구한다.

3.2 구 현

아래 [그림 3.2]는 Nucleus-Wrapper 상에서 태스크의 생성 시간 즉 Time-latency 를 구하는 코드이다.

```
/*target API code*/
Int taskSpawn(…)
{
        /*default overhead of get_usec() */
        before = get_usec();
        after = get_usec();
        default = after - before;

        start = get_usec();
        /* code lines of taskSpawn() */

        stop = get_usec();
        duration = stop - start;
        delta_time = duration - default;

        printf( "…%ld…" ,delta_time);
        return STATUS;
}
```

[그림 3.2] Nucleus-Wrapper 에서의 Time-latency 측정

아래 [그림 3.3]은 iRTOS™ 에서 태스크의 생성과 삭제에 대한 시간지연(Time-latency)값을 구하는 코드이다.

같은 방법으로 각 메세지 큐와 세마포어의 생성 및 삭제에 대한 시간지연(Time-latency)값을 구하였다.

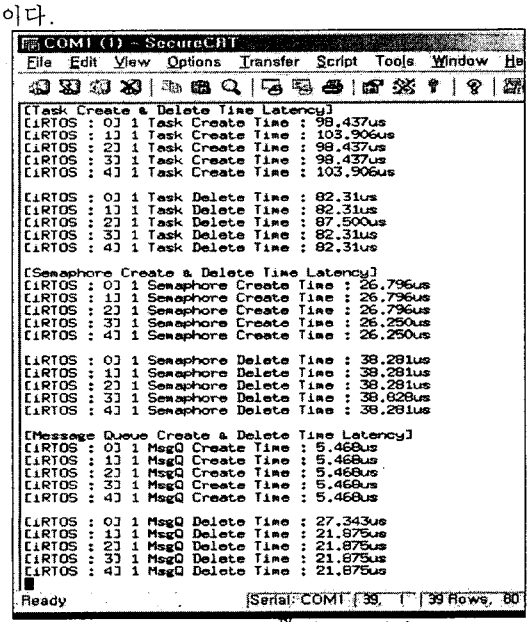
```
for( j = 0; j < 10; j++) {
  sum = MK_GetSystemClock();
  for( i = 0; i < 1000; i++) {
    MK_GetMemory(&gSystem_mem,&pAddr,128,MK_NO_WAIT);
    MK_CreateTask(&TestTCB[i], Test1, 20,
                  "TestFunc", 5, pAddr, 128, 1, 0);
  }
  sum = MK_GetSystemClock() - sum;
  sum = sum * 1400000 / 256;
  create_sum[j] = sum;

  sum = MK_GetSystemClock();
  for( i = 0; i < 1000; i++)
    MK_DeleteTask(&TestTCB[i]);
  sum = MK_GetSystemClock() - sum;
  sum = sum * 1400000 / 256;
  delete_sum[j] = sum;
}
```

[그림 3.3] iRTOS™ 에서의 Time-latency 측정

4. 태스트 환경 및 결과

ARM920T CPPU 기반의 S3C2440A 보드를 사용하여 실시간 운영체제 iRTOS™ 와 Nucleus-Wrapper 에서 시간지연 값을 측정하기 위한 방법을 커널 수준에서 설계하고 구현 및 테스트하였다. 아래 [그림 4.1]은 iRTOS™ 에서 시간 지연(Time-latency)에 대한 테스트 결과로 μs 단위로 출력하였으며 Nucleus-Wrapper 도 같은 방식으로 결과를 얻었다. [표 4.1]은 iRTOS™ 와 Nucleus-Wrapper 에서 측정한 Time-latency 의 평균값을 비교한 것이다.



[그림 4.1] iRTOS™ 테스트 결과

[표 4.1] Time-latency 비교

함수	iRTOS	Nu-Wrapper	Time latency
Task Create	100.62	103.9	3.28
Task Delete	83.35	90.5	7.15
MagQ Create	5.47	7.2	1.73
MsgQ Delete	22.97	26.21	3.24
Semaphore Create	26.58	32.2	5.62
Semaphore Delete	38.39	42.6	4.21

위의 결과값에서 미들웨어를 사용하였을 경우 Time-latency 가 1/20 정도의 증가한다. 1/20 정도 오버헤드가 발생하지만 이식성을 고려할 경우 큰 문제가 되지 않는다는 것을 알 수 있다.

5. 결론 및 향후 과제

본 논문에서는 iRTOS™ 와 Nucleus-wrapper 상에서의 태스크, 메시지 큐, 세마포어의 생성/삭제에 대한 시간 지연(Time-latency)을 측정 및 비교 하였다.

결과적으로 Wrapper 와 같은 미들웨어를 사용하더라도 오버헤드 차이가 미세하다는 것을 확인할 수 있었으며 이런 결과는 미들웨어 개발의 필요성과 이미 넓게 퍼져있는 국외 실시간 운영체제의 환경속에서 iRTOS™ 기반의 미들웨어를 사용함으로써 국내 iRTOS™의 확장을 기대해 볼 수 있다.

향후, 본 연구의 성과를 바탕으로 실용성 있는 미들웨어를 설계하고 구현하고 국산 iRTOS™ 에 적합한 많은 미들웨어 프로그램의 개발이 진행되어야 한다.

참고 문헌

[1] S3C2440A 32-bit RISC Microprocessor user's Manual Reivision 0.12

[2] iRTOS User's Guide, 2003

[3] Jean, J.Labrosse, "*uC/OS The Real-Time Kernel*", R&D Publications, 1995

[4] Accelerated Technology, Nucleus PLUS Internals Manual, 1996.

[5] 안희중, 백대현, 성영락†, 이철훈 "*Design and Implementation Real-Time Operating Systems for DVD Players*", 한국정보과학회 추계학술발표논문집, vol.3, no.2-1, pp.340-342, Oct.2003