크로스 컴파일러에서의 효율적인 메모리 사용 기법에 대한 연구

A Study for the Efficient Memory Management in time of using Cross Compiler

삼성전자, 디지털 미디어 연구소, Software Platform Lab.
경보현, 전승훈

Kyong Bo Hyun, Jeon Seung Hun
Software Platform Lab., Digital Media R&D Center, Samsung Elec.
e-mail) kyonggo@samsung.com

요 약

본 논문은 RTOS(Real-Time Operation System, 리턴어드레스를 위한 유저스택사용 RTOS가 탑재된 CE(Consumer Electronic)제품상에서 리턴어드레스가 유저스택으로 저장하는 것을 지원하지 않는 컴파일러를 위한 알고리즘이며 실험을 위하여 제안된 알고리즘을 상용 컴파일러에 적용하여 비교해보도록 하겠다. 우선 기존 컴파일러 알고리즘으로는 Task마다 할당된 유저스택영역이 존재하며 Task가 수행중 발생된 리턴어드레스는 즉시 할당된 유저스택으로 저장하는 알고리즘을 갖고있다. 이런 알고리즘으로 인하여 인스트럭션이 수행중 빈번한 메모리 접근(external memory)가 발생한다. 그러나 제안된 알고리즘은 Task 수행중에는 리턴어드레스를 시스템스택(internal memory)에 저장한 후 Task 전환이 발생할 경우 일시에 시스템 스택에 저장된 리턴어드레스를 유저스택으로 이동하게 되므로 Task 수행중에는 시스템 스택만을 접근하므로 task의 수행시간을 단축할 수가 있다. 그리고 실험을 위하여 상용 컴파일러들에 본 알고리즘을 적용하였다. 상용 컴파일러로는 매번 리턴어드레스를 자동으로 Task별 할당된 유저스택에 저장할 수있도록 지원해주는 TASKING 컴파일러(Altium 사)와 그렇지 않은 KEIL컴파일러(KEIL사)가 있으며 본 알고리즘을 KEIL 컴파일러에 적용하여 실험을 하여 TASKING 컴파일러와 비교한 결과 유저스택을 지원하는 TASKING(Altium사) 컴파일러에서 구현한 CE제품의 Response time이 KEIL 컴파일러에서 구현한 CE제품의 Response time 값이 같게 나왔다. 그러므로 KEIL 컴파일러상에 본 알고리즘을 적용시킬 경우 RTOS가 탑재된 CE제품을 보다 용이하게 구현할 수가 있다.

Ⅰ. 서 론

임베디드 시스템을 개발할 때 일반적으로 호스트와 타겟이란 말을 사용하는데 여기서 호스트란 임베디드 시스템을 개발하는 과정에서 프로그램을 개발하는 컴퓨터를 가리킨다. 예를 들어 C166 프로세서로 하드웨어를 꾸미고 여기에 Nucleus 를 OS 로 사용하는 프로젝트를 생각해보자. 처음 하드웨어를 만들어 커널을 올리는 작업을 할 때 만들어진 하드웨어엔 아무 프로그램도 올라가 있지 않기 때문에 다른 곳에서 커널을 만들어 하드웨어에 심어줘야한다. 그리고 Nucleus 커널을 만들어주는 시스템을 호스트라 고 하며 보통은 PC에서 개발해 이식할 것이므로 PC가 호스트가 되겠다. 그리고 임베디드 시스템의 개발에서 만들어지는 하드웨어를 타겟이라한다. 호스트로 Athlon CPU 를 사용하는 PC를 사용하고 있다. 여기에서 실행되는 gcc 는 386, 486,

586, K6, 686 을 지원한다. 그러나 임베디스 시스템에 사용되는 프로세서는 지원하지 않고 있다. 그러므로 프로그램을 컴파일해도 실제 임베디드 시스템에 사용된 프로세서에서는 실행할 수 없게된다. 그러므로 실행은 호스트에서 되지만 만들어지는 코드는 타겟 시스템에서 돌아갈 수 있는 컴파일러가 필요해 진다. 이 것이 바로 크로스 컴파일러다.[4]

본 논문의 구성은 2.1 장에서 TASKING 컴파일러 리턴 어드레스 기법에 대해서 알아보고 2.2 장에서는 KEIL 컴파일러 리턴어드레스 기법과 제안된 알고리즘에 대하여설명을 하며 3 장에서는 제안된 기법에 대한 실험 결과를 제시하도록하며 마지막으로 4 장에서는 결론을 맺도록하겠다.[1][2]

II. 본론

본 논문은 RTOS(Real-Time Operation System, 리턴어드레스를 위한 유저스택 사용 RTOS)가 탑재된 CE(Consumer Electronic)제품상에서 리턴어드레스가 유저스택으로 저장하는 것을 지원하지 않는 컴파일러를 위한 알고리즘이다.

2.1 TASKING 컴파일러 Return address 기법

임베디드 소프트웨어는 임의의 시간에 발생하는 다양한 이벤트에 대응하여야 한다. 디스플레이용 임베디드 소프트웨어에서 예를 들면, IR(infrared remote control) 의 키를 눌렸을 때, Vsync 가 system input 으로 detect 되었을 때, OSD(On Screen Display)를 디스플레이할 필요가 있을 때 , serial communication device 에서 transmit data request 가 발생했을 때, 그리고 time delay 가 expire 할 때 등등이 있다. 가장 중요한 이벤트로써 하드웨어 인터럽트와 같은 외부 이벤트가 발생했을 경우가 있다. 대부분의 경우에는 인터럽트 서비스 루틴이 활성화 되어 각각의 interrupt signal 에 해당하는 서비스를 수행하지만, 모든 경우에 대응하기에 복잡하고 오버헤드가 발생하므로 task 구조가 필요하게 되며 이와 같은 이유로 RTOS 를 적용해야 보다 구조화된 임베디드 소프트웨어를 만들 수 가있다.[3]

상기와 같은 이유로 Nucleus RTOS(이하 RTOS)를 임베디드 소프트웨어에 적용하기 위하여 TASKING 이라는 크로스-컴파일러를 채택할 수가있다. 그리고 Task 의 수행시간을 빠르게 하기위하여 TASKING 컴파일러는 task 수행중 리턴 어드레스를 유저스택에 저장하도록하는 기능을 지원해준다. 이런 기능으로 인하여 표 1 과 같이 Nucleus 에 대한 각 API 들과 Context Operation 에 대한 성능측정표가 측정되었다. [1]

Processor	Clock Speed (Mhz)	Pre-scaler	Date
C166	33.33	32	2003-07-11

Variable	Raw Clock Count	Adjusted Clock Count	Time (microsec)
T_Bench	14	N/A	13.4
T_Create_Task_IC	289	275	264.0
T_Delete_Task_IC	112	98	94.1
T_Suspend_Task_IC	144	130	124.8
T_Relinquish_IC	86	72	69.1
T_Create_Mailbox_IC	152	138	132.5
T_Delete_Mailbox_IC	152	138	132.5
T_Send_To_Mailbox_IC	116	102	97.9
T_Receive_From_Mailbox_IC	117	103	98.9
T_Create_Queue_IC	226	212	203.5
T_Delete_Queue_IC	158	144	138.3
T_Send_To_Queue_IC	156	142	136.3
T_Receive_From_Queue_IC	162	148	142.1

Type of Context Operation	Start	End	Time (microsec)
Idle Save	55141	55162	20.2
Thread Save	55147	55186	37.4
Idle Restore	55204	55220	15.4
Thread Restore	55229	55245	15.4
Interrupt Latency (MIN)	N/A	N/A	15.4
Interrupt Latency (MAX)	N/A	N/A	37.4

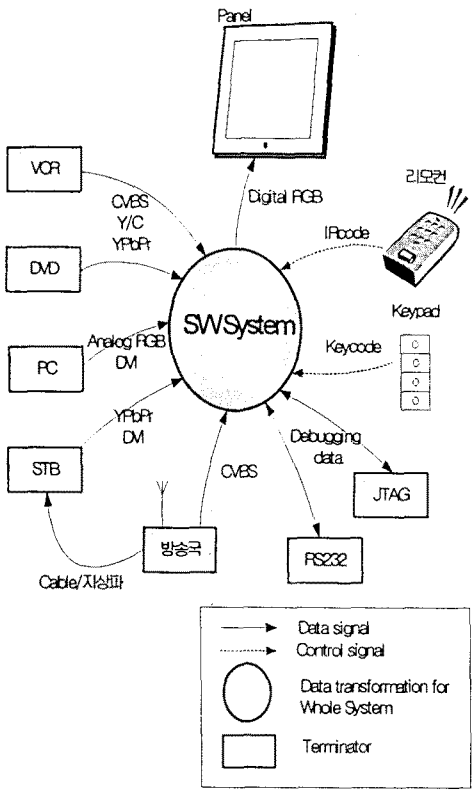
표 1 TASKING 컴파일러 적용 측정치

2.2 KEIL 컴파일러 Return address 기법

2.2.1 기존 KEIL 컴파일러

RTOS 를 그림-1 과 같은 CE 시스템에 적용하기 위하여 KEIL 컴파일러라는 Cross-compiler 를 선택하였을 경우 발생되는 문제점들을 생각해보자. 우선 RTOS(Nucleus)는 Task 처리 속도를 높이기 위하여 Task 수행시 발생되는 리턴 어드레스를 시스템 스택이 아닌 유저스택으로 보관 되어지도록 구현되어져 있다. 그러나 KEIL 컴파일러는 RTOS 탑재 CE 제품 개발시 최적으로 컴파일러로는 볼수가 없으며 리턴 어드레스 처리 부분에 대해서는 발생할 때마다 시스템 스택에 보관을하는 기법을 채택하였다. 일반적으로 Super loop 의 특성를 갖는 Firmware 에서는 기본적으로 리턴 어드레스가 발생할때마다 시스템 스택을 사용하도록 되어져 있으며 KEIL 컴파일러는 이런 부분을 볼 때 RTOS 에 적합한 것이 아니라 Super loop 형태를 갖는 Firmware 구

조에 적합하다고 볼 수가있다. 그러므로 Super loop Firmware 구조에 KEIL 컴파일러를 사용하면서 RTOS를 적용시킬 경우 앞에서 설명한 문제에 의하여 CE 제품구현이 어려울 수가 있다. 그러나 본 논문에서 제시되는 알고리즘 기법을 사용한다면 이런 어려움은 쉽게 해결할 수 가있다.



[2]

그림 1 CE 용 디스플레이 시스템

2.2.2 제안된 알고리즘이 적용된 KEIL 컴파일러

앞에서 설명했 듯이 Super loop Firmware 개발에 적합한 KEIL 컴파일러에 RTOS 를 적용할 경우에 아래와 같은 알고리즘으로 CE 제품개발시 유용함을 볼수가 있다.

그림 2.2 에서 보는바와 같이 우선순위 정책상 Priority 가 작은 값을 우선순위가 가장 높다고 하자. 그러면 TASK 1 이 가장 먼저 수행이될 것이다. 그리고 수행중 발생되는 리턴 어드레스는 임시로 C166 내부메모리에 존재하는 시스템 스택에 쌓일 것이다. 그리고 태스크 전환이 발생하면서 그 다음 우선순위 값을 갖고있는 TASK 3 이 수행될 것이다. 본 알고리즘은 TASK 3 이 수행되기 전에 TASK 1 이 수행할 때 발생된 리턴 어드레스를 모두 TASK 1 이 할당받은 유저 스택으로 이동한다. 이런 과정속에 다시 시스템 스택은 비어지게 되면 TASK 3 는 전용으로 시스템 스택을 사용할 수가 있게 된다. TASK 3 의 수행이 끝났을때도 앞 과정과 동일한 과정으로 시스템스택의 데이터를 지금까지 수행되고 있었던 TASK(TASK 3)의 유저스택에 모두 이동한다. 그리고 본 알고리즘 적용으로 인하여 KEIL 컴파일러 상서에 RTOS 를 보다 쉽게 사용할 수가 있으며 이와 함께 Task 수행중 시스템 스택에만 접근하므로 Task 에 대한 수행시간도 높일 수 있는 장점도 발견했다.

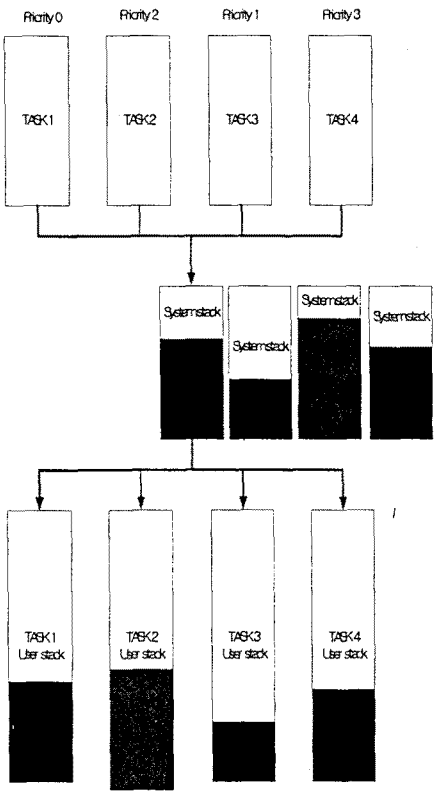


그림 2 제안된 알고리즘

그리고 본 알고리즘을 KEIL 컴파일러에 적용하여 RTOS 의 API 들의 응답시간을 측정한 결과 표-2 와 같은 결과를 얻었다.

Processor	Clock Speed (Mhz)	Pre-scaler	Date
C166	33.33	32	2003-07-11

Variable	Raw Clock Count	Adjusted Clock Count	Time (microsec)
T_Bench	14	N/A	13.4
T_Create_Task_IC	285	271	260.2
T_Delete_Task_IC	86	72	69.1
T_Suspend_Task_IC	135	121	116.2
T_Relinquish_IC	69	55	52.8
T_Create_Mailbox_IC	121	107	102.7
T_Delete_Mailbox_IC	130	116	111.4
T_Send_To_Mailbox_IC	101	87	83.5
T_Receive_From_Mailbox_IC	100	86	82.6
T_Create_Queue_IC	202	188	180.5
T_Delete_Queue_IC	135	121	116.2
T_Send_To_Queue_IC	118	104	99.8
T_Receive_From_Queue_IC	123	109	104.7
	0		

Type of Context Operation	Start	End	Time (microsec)
Idle Save	55142	55166	23.0
Thread Save	55147	55215	65.3
Idle Restore	55207	55224	16.3
Thread Restore	55256	55272	15.4
Interrupt Latency (MIN)	N/A	N/A	15.4
Interrupt Latency (MAX)	N/A	N/A	65.3

표 2 KEIL 컴파일러에 제안된 알고리즘 적용측정치

III. 실험결과

본 제안된 알고리즘은 33.3 MHz 의 CPU 와 TASKING, KEIL tool 에서 실험하여 아래와 같은 비교결과을 얻을 수가 있었다.

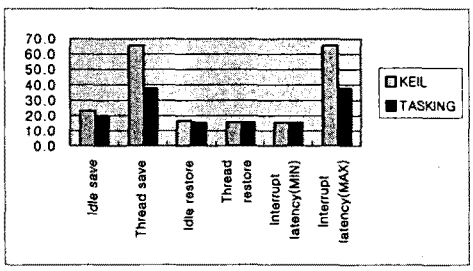


그림 3 Context Operation 비교

그림 -4 는 Context operation 을 비교하였으며 그림-5 는 RTOS 에서 사용되어지는 API 성능을 측정하여 KEIL 과 TASKING 를 비교한것이다.

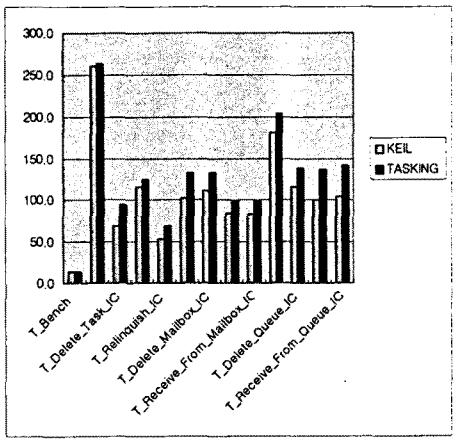


그림 4 RTOS function 비교

IV. 결 론

본 논문은 RTOS 가 탑재된 CE(Consumer Electronic)제품상에서 리턴 어드레스가 유저스택으로 저장하는 것을 지원하지 않는 컴파일러를 위한 알고리즘이며 실험을 위하여 제안된 알고리즘을 상용 컴파일러에 적용하여 비교해본 결과 유저스택을 지원하는 TASKING(Altium 사) 컴파일러에서 구현한 CE 제품의 Response time 이 KEIL 컴파일러에서 구현한 CE 제품의 Response time 값이 유사하게 나왔다. 그러므로 KEIL 컴파일러상에 본 알고리즘을 적용시킬 경우 RTOS 가 탑재된 CE 제품 용이하게 구현 할 수가있었으며 Task 내 수행시간도 줄일 수 있는 장점도 보였다.

[참 고 문 헌]

[1] KEIL Inc. , "166/167 Assembler and Utilities User' s Guide 07.96 " ,Doc, p.1-352, 2000.

[2] Altium Inc. , "C166/ST10 v7.5 C CROSS- COMPILER USER' S GUIDE" , Doc., p.1-551, 2001.

[3] Samsung Electronics Inc., Seung-Hun Jeon, " R2D2-SWHDS 2003.03.05 Ver. 1.00 ", pp.40-59, March, 2003

[4] Accelerated Technology, Inc. " Nucleus PLUS Reference Manual 0001026-001 Rev.103" , 2001