技術解說

타겟 머신(Target Machine)에서의 O. S. 디버깅(Debugging)

박 치 항
한국전자기술연구소 시스템부 선임연구원

Ⅰ. 배 경

컴퓨터 시스템 개발 계획[3]에 발맞추어 O. S.를 개발, 이전(porting) 및 수정하는 작업(이하 총괄적으로 개발 작업으로 단일화하여 언급하려 함)에 대해 본격적으로 연구할 때가 온것 같다. 이러한 작업을 하기에 앞서 개발 작업의 한 부분인 디버깅(debugging)에 대해 언급하려 한다.

O. S. 디버깅은 두 단계로 분류할 수 있다. 첫 단계는 개발 도구로서 호스트(host) 컴퓨터가 필요하며 호스트 컴퓨터의 여러 가지 소프트웨어, 즉 O. S.와 디버깅 도구(debugging tool) 그리고 크로스 컴파일러(cross compiler), 크로스 어셈블러(cross assembler) 및 시뮬레이터(simulator) 등의 크로스 소프트웨어를 이용한 디버깅 과정이고 두번째 단계는 개발된 O. S.가 최종적으로 담길 타겟 머신(target machine)에서의 디버깅 과정이다.

타겟 머신에서의 디버깅 방법에 대해서는 컴퓨터 시스템 제조회사마다 고유의 것이 존재하리라 생각되지만 실제로 알려진 것은 별로 없어 "lost worlb"[2]로 불리워지고 있다. 하지만 마이크로컴퓨터가 활발하게 이용되면서부터 특히 타겟 머신에서의 O. S. 디버깅 방법에 대한 연구가 요구되어 본문에서는 디버깅의 한 방법으로서 세 가지 종류의 디버거(degubber)를 소개하고[1] 이를 모델로 하여 O. S. 디버깅 도구 설계시의 문제점 및 요구 조건을 분석하여 타겟 머신에서의 O. S. 디버깅시 발생하는 어려운 문제를 부분적이나마 해결하는데 도움이 되고져 한다.

Ⅱ. O. S. 디버깅의 문제점

호스트 컴퓨터에서의 O. S. 디버깅은 일반적인 프로그램의 디버깅과 크게 다를 바가 없다. 하지만 O. S.는 그 자체가 하드웨어와 밀접한 관계를 가지기 때문에 호스트 컴퓨터에서 디버깅하기 곤란한 부분, 즉 타겟 머신 고유의 기능, 인터럽트(interrupt), 타이밍(timing) 사이징(sizing), 애큐러시(accuracy) 그리고 디바이스 드라이버(divice driver)등에 관한 부분이 존재한다. 따라서 이러한 부분에 대해서는 어쩔 수 없이 타켓 머신에서 디버깅해야 하는데 개발된 O. S.가 담길 타겟 머신은 일반적으로 소프트웨어가 존재하지 않는 베어 머신(bare machine)이다. 따라서 어떠한 디버깅 도구도 존재하지 않는다. 하지만 복잡하고 방대한 O. S.를 디버깅한다는 것은 거의 불가능에 가까운 일이라고 할 수 있다.

Ⅲ. O. S. 디버깅 도구(Debugging Tool)

타겟 머신에서의 O. S. 디버깅시 첫번째로 부딪히는 문제가 디버거가 존재하지 않는다는 사실이다. 최근에 생산되고 있는 마이크로나 미니컴퓨터에는 ROM에 디버거 기능이 있어 타겟 머신에서의 디버깅에 크게 도움을 주고 있다. ROM에 기억된 디버거는 사용하기가 쉬워 매우 편리하지만 하드웨어가 별도로 필요하게 된다. 새로 개발되어 일차적으로 호스트 컴퓨터에서 테스트가 끝난 프로그램(여기서는 O. S.)은 디스크(disc)에 옮겨진 후 타겟 머신에서는 다시 이 디스크에 있는 내용을 읽어 들여야 하는데 이때 디스크의 종류에 따라 드라이버 루틴(driver routine)이 달라져야 하기 때문에 ROM에 이러한 점을 모두 반영하기가 쉽지 않다. ROM에 있는 디버거를 사용하는 외에 호스트 컴퓨터에서 사용되고 있는 디버거를 수정하여

사용하는 방법도 생각해 볼 수 있겠으나 타겟 머신이 베어 머신이기 때문에 메모리(memory)에 로드(load)하는 과정이 어렵다. 이상과 같은 불편한 점을 해결하기 위하여는 O. S. 자체가 자신의 디버깅을 위한 고유의 디버거를 가지게 하는 방법을 사용할 수 있다. (자세한 설명은 본문 Ⅲ. 2 참조) 이 방법을 사용하면 부트스트랩(bootstrap) 과정을 통해 디버거가 쉽게 로드(load)될 수 있을 뿐만 아니라 디버깅을 몇 단계로 분류하여 각 단계별로 디버깅이 가능하고 O. S. 고유의 기능을 테스트할 수 있도록 디버거를 설계하여 개발 과정에서의 디버깅 뿐만 아니라 O. S. 유지보수(maintenance)에서도 편리하게 사용될 수 있다. 본문에서는 O. S. 자체의 디버깅을 위해 몇 개의 디버거가 필요한 이유를 디버거 설계시의 고려 사항을 통해 알아보고 이러한 고려 사항을 기본으로 하여 디버거의 적정수를 정하고 디버깅 과정을 설명한 후 각 단계별 디버깅에 필요한 디버거의 특색 및 기능에 대해 언급하고져 한다.

1. 디버깅 도구 설계시 고려 사항

O. S. 자신을 디버깅하기 위한 도구를 설계하기에 앞서 다음과 같은 사항을 고려해야 한다.

i) 에러(error)는 부투스트랩(bootstrap) 초기부터 발생할 수 있다.

ii) 디스크 블럭(disc block) 0는 딴 블럭(block)보다 읽기 쉽다.

iii) 디스크 블럭 0만 읽는것은 블럭 0부터 2블럭 이상 읽는 것보다 간단하다.

iv) 디버거가 편리하게 사용되기 위하여는 가능한한 많은 기능을 소유하는 것이 좋다.

v) O. S. 의 레지던트(resident) 부분에 대한 디버거는 O. S. 루틴(routine)의 도움없이 동작해야 한다.

vi) 디버거의 기능이 다양해지고 O. S. 고유의 기능을 테스트 하기 위해서는 O. S의 도움을 받아야 한다.

이와 같은 요구 조건을 고루 만족시키는 디버거를 설계하려면 디스크 블럭 0에 디버거를 존재시키고 O. S.의 인스톨레이션(installation) 과정을 몇 단계로 분류하여 각 단계별로 서로 다른 디버거가 존재하게 할 필요를 느낀다.

2. 디버깅 도구의 적정수 및 디버깅 과정

지금까지 O. S. 디버깅을 위하여 디버깅 과정을 몇 단계로 나누는 것이 바람직하다는것을 살펴보았다. 그러면 실제로 몇 단계로 나누는 것이 이상적인가를 검토하기 위해 O. S.를 메모리 로딩(loading) 과정과 연관지워 다음과 같이 세 부분으로 나눌 수 있다.

블럭 0 블럭 1 블럭n-1 블럭n

*IPL부분	메모리 레지던트 (memory resident)	디스크 레지던트 (disc resident)

디스크에 담긴 O. S. 의 구성

*IPL(initial program loading)

O. S.를 위와 같이 세 부분으로 구분하고 이에 대응하는 세 개의 별도의 디버거를 설정하여 해당 부분에 포함시켜 각각 I-디버거, R-디버거, D-디버거라고 하면 디버깅 과정은 다음과 같이 진행된다.

i) 부트스트랩을 위한 첫단계인 블럭 0 (IPL부분)의 로드(load)는 보통 컴퓨터에서 볼 수 있는 스위치나 ROM 루틴(routine)에 의한다.

ii) 일단 블럭 0가 로드(load)되면 이후의 디버깅은 I 디버거에 의해 행해진다.

iii) 매모리 레지던트(memory resident) 부분의 로드(load)는 IPL 부분의 도움으로 행해진다.

iv) I 디버거는 메모리 레지던트(memory resident)의 로드(load)가 끝나고 R디버거가 로드(load)될 때까지 사용된다.

v) R디버거는 O. S.의 도움없이 단일 사용자 모드(single user mode)에서 모든 메모리 레지던트(disc resident)와 디스크 레지던트(disc resident) 부분에 대한 디버깅에 사용된다.

vi) O. S. 커널(kernel) 부분에 대한 테스트가 끝나면 D디버거를 사용할 수 있는데 이 디버거는 O. S. 루틴(routine)을 사용하여 다중 사용자 모드(multi user mode)에서 사용할 수 있다.

3. 디버깅 도구의 특색 및 기능

앞 두절에서 살펴 본 디버깅 도구 설계시의 고려 사항을 단계별로 설정된 각 디버거에 적용시키면 다음과 같은 특색 및 기능을 얻게 된다.

1) I-디버거

블럭 0에 존재하기 때문에 크기에 제한을 받아 디버깅을 위한 최소한의 기능만을 보유하게 되며 간단한 디스크 드라이버(disc driver)를 포함한 IPL 루틴(routine)이 필요하다. 디스크 드라이버 루틴(disc driver routine)은 디스크 또는 콘트롤러(disc controller)에 따라 다르며 디스크 입출력은 로지컬 블럭 어드레스

(logical block address)를 사용하여 행해진다.

디버거를 편리하게 사용하고 메모리에 로드(load)된 O. S.의 영향을 받지 않기 위하여는 포지션 인디펜던트(position independent)로 작성되어야 한다. 디버거 명령(duebugger command)을 기준으로한 기능은 다음과 같다.

i) Display memory
ii) Store number into memory
iii) Move block in memory
iv) Disc i/o
v) Start IPL sequence

2) R-디버거

기능면에서 I-디버거의 수퍼셋(superset)이며 크기에 제한 받지 않는다. O. S. 커널(Kernel) 또는 메모리 레지던트(memory resident)에 대한 디버깅에 사용되기 때문에 O. S. 루틴(routine)의 도움없이 단일 사용자 모드(single user mode)에서 사용해야 한다. I-디버거의 디스크 드라이버 루틴(disc driver routine)을 그대로 사용할 수 있으며 디스크 입출력은 I-디버거와 마찬가지로 로지컬 블럭 어드레스(logical block address)를 사용한다. O. S. 의 메모리 사용에 대한 영향을 받지 않기 위하여 포지션 인디펜던트 코드(position independent code)로 작성 되어야 한다. 디버거 명령(debugger command)을 기준으로 한 기능은 다음과 같다.

i) Display register/flag
ii) Change register/flag contents
iii) Insert/remove breakpoint
iv) Trace
v) Dump memory
vi) Input/output ASCII string
vii) Jump
viii) List program in octal and symbolic
ix) Search

3) D-디버거

기능면에서 R-디버거의 수퍼셋(superset)이며 디버깅 하고져 하는 오퍼레이팅 시스템(operating system)이 일단 안정된 후에 사용 가능하다. 왜냐하면 이 디버거는 O. S. 루틴(routine)을 사용하여 다중 사용자 모드(multi user mode)에서 사용하기 때문이다. 로지컬 블럭 어드레스(logical block address)를 통한 디스크 입출력은 물론 파일(file) 시스템을 통한 디스크 액세스(disc access)가 가능하기 때문에 주로 파일(file)에 대한 디버깅에 편리하게 사용된다. 디버거 명령(debugger cammand)을 기준으로 R-디버거에 없는 새로운 기능은 다음과 같다.

i) Disc access using file name
ii) Modification of file information
iii) Access swap file
iv) 기타 시스템 고유의 기능에 대한 여러 가지 디버깅 기능

Ⅳ. 결 론

이상에서 살펴 본 단계별 디버거에 의한 디버깅 방법이 O. S. 디버깅에 편리하고 효과적으로 사용될 수 있지만 실제시행(implementation)을 위하여는 구체적인 연구가 필요하다. 인터럽트 처리 루틴(interrupt processing routine)의 디버깅은 여전히 어려운 문제의 하나이며 D-디버거를 사용하여 O. S.고유의 기능을 테스트 하기 위하여는 O. S. 설계시에 이 디버거에 대한 여러 가지 기능이 이미 반영되어 있어야 한다.

참 고 문 헌

[1] *IRIS 7. 3 Manager Reference Manual*. POINT 4 Data Co.

[2] Robert L. Glass, "Real-time : The 'lost world' of software debugging and testing", *Comm. of ACM*, vol. 23, no 5. May 1980.

[3] 전길남, "컴퓨터 시스템 개발," 전자공학회잡지 vol. 8, no. 4, Dec. 1981.

略語解說

SIC (Semiconductor integrated circuit 半導体 集積回路, 電氣·電子回路)

1또는 2이상의 半導体基板에 꾸며 넣은 回路素子를 상호 접속한 集積回路를 말한다.