논•문 16-3-4

불완전 연결 프로그램 실행을 위한 예외 처리 기법 (Exception Handling for Executing Incompletely Linked Programs)

한 경숙, 표 창우
(주)참좋은인터넷, 홍익대학교 컴퓨터공학과
khan@sogood.co.kr, pyo@hongik.ac.kr

요 약

응용 프로그램 개발자는 편집, 컴파일, 링킹, 다운로드, 테스팅 과정을 반복하며 프로그램을 개발한다. 이러한 개발 과정에서 프로그램의 초기 테스트를 가능하게 하고 프로그램 처리 과정의 과부하를 줄이기 위하여 점진적 링킹 방법을 사용할 수 있다. 프로그램 개발 초기 단계에 테스팅하거나 프로그래머가 다른 프로그래머의 모듈과 연결하지 않은 상태에서 테스팅 할 수 있기 위해서는 불완전 연결 프로그램의 실행이 가능해야 한다. 내장형 응용 프로그램 개발자의 경우 시스템이 완성되기 전에 개별적인 프로그램에 대한 테스팅을 필요로 하는 경우가 많으므로 불완전 연결 프로그램의 실행 방법이 유용하게 사용된다. 또한 불완전 연결 프로그램의 실행 과정에서 정의되지 않은 심벌에 대한 참조가 발생하더라도 시스템의 안정성을 유지해야 한다. 불완전 연결 프로그램을 테스팅 하는 경우 재배치가 이루어지지 않은 심벌의 집근으로 인하여 문제가 발생할 수 있다. 이 문제의 해결을 위하여 정의되지 않은 심벌을 예외처리 함수로 연결하여 부적절한 메모리 집근을 방지하였다. 예외처리 함수는 프로그램을 안정적으로 종료시키며 사용자에게 예외 상황 발생 원인을 보고하는 기능을 수행한다.

1. 서론

응용 프로그램 개발자는 편집, 컴파일, 링킹, 다운로드, 테스팅 과정을 반복하며 프로그램을 개발한다. 이러한 개발 과정에서 프로그램의 실행은 참조하는 모든 외부 심벌에 대한 연결이 완료되어야 가능하므로 관련된 모듈이 모두 작성된 후에 테스트하게 된다. 프로그램의 외부 심벌이 완전히 연결되지 않은 상태에서 실행할 수 있게 되면 일부 모듈만 작성한 상태에서의 실행이 가능해지므로 개발 초기 단계에서 테스트가 가능하다. 또한 프로그램 개발을 진행하면서 작성되는 모듈을 효과적으로 연결하기 위하여 점진적으

로 연결해 나가는 방법이 필요하다. 점진적 링킹 방법은 프로그램 수정 과정에서 반복적으로 발생하는 링킹작업의 과부하를 감소시킬 수 있다[1].

프로그램 개발 초기 단계에 테스팅 하거나 프로그래머가 다른 프로그래머의 모듈과 연결하지 않은 상태에서 테스팅 할 수 있기 위해서는 불완전 연결 프로그램의 실행이 가능해야 한다. 또한 불완전 연결 프로그램의 실행 과정에서 정의되지 않은 심벌에 대한 참조가 발생하더라도 시스템의 안정성을 유지해야 한다. 불완전 연결 프로그램의 실행이 가능해지면 개발의 초기 단계에서의 테스트가 가능하며 테스팅 대상을 축소할 수 있다. 따라서 오류가 발생하는 경우 디버깅이 용이하다. 이러한 방법은 내장형 시스템을 위한 응용 프로그램 개발 환경에서 유용하게 사용될 수 있다. 내장형 응용 프로그램 개발자의 경우 시스템이 완성되기 전에 개별적인 프로그램에 대한 테스팅을 필요로 하는 경우가 많으므로 불완전 연결 프로그램의 실행 방법이 유용하게 사용된다.

호스트 단독 개발 환경에서의 컴파일, 링킹, 실행 과정을 내장형 응용 프로그램 개발에 그대로 적용하는 경우 프로그램 수정 과정의 오버헤드가 발생한다. 내장형 응용 프로그램 개발 환경은 네트워크로 연결되어 있는 호스트 시스템과 타겟 시스템으로 이루어진다. 프로그램 개발은 자원이 풍부한 호스트 시스템에서 이루어지며 내장형 프로세서가 탑재된 타겟 시스템으로 다운로드 되어 실행되게 된다. 이러한 개발 환경에서 프로그램의 일부만 수정되거나 적은 부분이 추가되는 경우에도 프로그램 전체가 다시 링킹, 다운로드 되어야 하므로 불필요한 추가 비용이 발생한다.

불완전 연결 프로그램을 실행하는 경우 재배치가 이루어지지 않은 심벌의 접근으로 인하여 문제가 발생할 수 있다. 재배치가 이루어지지 않은 명령을 수행하면 데이터 접근이나 프로그램 제어 과정에서 접근할 메모리 영역을 예측할 수 없다. 프로그램 실행 시간에 부적절한 메모리 접근에 의해 프로그램 실행이 중단되거나 시스템 오류가 발생하게 되면 시스템을 신뢰할 수 없게 된다. 이러한 문제를 해결하기 위하여 정의되지 않은 심벌에 대한 예외 처리 방법을 제공한다. 예외처리 함수는 프로그램을 안정적으로 종료시키며 사용자에게 예외 상황 발생 원인을 보고하는 기능을 수행한다.

2. 관련연구

Simon은 호스트와 타겟이 동일한 단일 시스템 개발 환경의 링커와 내장형 응용 프로그램 개발 환경의 링커를 비교하였다[4]. 단일 시스템 개발 환경의 경우 링킹 과정에서 관련된 목적 모듈을 하나로 연결하여 실행 파일을 생성한다. 프로그램의 실행을 위해 로더(loader)가 메모리에 공간을 할당받아 적재하고 실제 메모리 주소로 변경하는 기능을 수행한다. 내장형 응용 프로그램 개발 환경에서는 링커가 타겟 메모리에 존재하는 모듈

에 대한 정보를 이용하여 연결 과정을 수행한다. 따라서 실제 적재된 타겟 메모리 주소를 직접 적용하여 연결을 수행하고 이를 위한 모듈들의 위치 정보를 유지한다.

Tornado는 호스트 시스템과 타겟 시스템을 네트워크로 연결하고 있는 개발 환경에서 프로그램을 편집, 컴파일, 디버깅하고 타겟에 적재하기 위한 도구들을 제공한다[3][4]. Tornado는 최근 점진적인 링킹 방법을 지원하여 프로그램 개발자가 프로그램의 일부 모듈을 적재하여 실행하기 위한 방법을 제공하였으나 예외 상황에 대한 처리 방법이 제공되지 않는다.

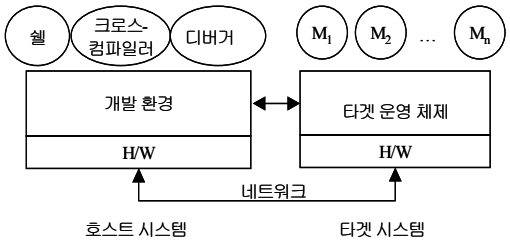
Quong은 단일 시스템 개발 환경에서의 점진적 링킹 방법을 제안하였다[6]. 프로그램을 구성하는 모듈 중 수정되는 모듈만을 다시 적재하고 링킹하기 위하여 다시 적재되는 모듈을 원래 적재되었던 위치에 적재하는 방법을 사용한다. 수정된 모듈의 크기가 증가하여 원래의 위치에 적재되지 못하는 경우를 위하여 모듈 사이에 일정 공간을 유지하나 그 공간으로 부족한 경우에는 전체 모듈에 대해 다시 링킹 작업을 수행해야 한다. 본 논문에서 사용하는 점진적 링킹 방법은 프로그램이 개발되는 시스템과 실행될 시스템이 분리되어 있다는 차이점이 있다. 또한 타겟 메모리에 적재되어 있는 모듈에 대해서도 원격으로 재배치 작업을 수행하므로 수정된 모듈을 원래의 위치에 적재할 필요가 없다.

단일 시스템 개발 환경에서 객체 지향 언어에 적합한 예외 처리 기능이 연구되었다[7]. 이 방법은 프로그램 자체에 예외 처리를 위한 코드를 삽입할 수 있는 기법을 제공한 것으로 예외 상황이 발생하면 지정해 놓은 블록을 빠져나가며 그 이후의 작업을 계속 수행한다. 본 논문에서 제공한 예외 처리 방법은 개발이 완료되지 않은 프로그램에 의해 실행 과정에 발생하는 예외 상황에 대한 처리 방법이다. 따라서 예외 상황이 발생하면 프로그램의 실행을 중단하고 사용자에게 예외 상황에 대해 보고함으로써 시스템의 안정성을 높이고 프로그램 개발자가 미완성된 프로그램 부분에 대해 인식하게 하는 목적이 있다.

3. 개발 환경

내장형 프로세서가 탑재된 타겟 시스템은 제한된 하드웨어와 소프트웨어를 사용한다. 내장형 시스템은 특정 목적을 위해 사용되는 시스템으로 일반적인 개발 도구나 풍부한 하드웨어를 지원하기 어렵다. 따라서 내장형 시스템을 위한 응용 프로그램 개발 환경은 소프트웨어와 하드웨어 자원이 풍부한 시스템에서 프로그램을 개발하도록 한다. 이 시스템을 호스트 시스템이라 한다.

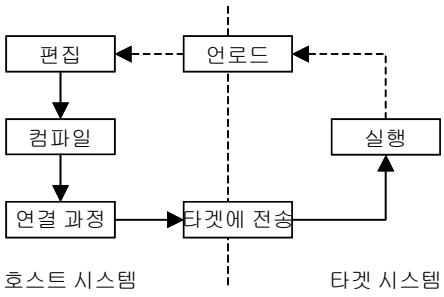
내장형 시스템을 위한 응용 프로그램 개발 환경은 [그림 1]과 같다. 호스트 시스템과 타겟 시스템은 네트워크로 연결되어 있으며 호스트 시스템에서 개발된 프로그램을 타겟 시스템으로 다운로드하여 실행한다. 내장형 시스템을 위한 응용 프로그램 개발 환경은 프로그램 개발이 이루어지는 호스트 시



[그림 1] 내장형 시스템을 위한 개발환경

스템에 개발을 위한 도구들이 포함된다. 프로그램 개발에 사용하는 컴파일러와 디버거, 실행과 타겟 시스템의 정보를 보기 위한 쉘과 같은 도구가 호스트 시스템에 존재한다. 호스트 시스템에 존재하는 미들웨어가 타겟 시스템과 이러한 도구들을 연결하여 도구에서 요청하는 작업을 타겟으로 전달하고 타겟의 정보를 도구로 전달하는 기능을 한다. 이러한 미들웨어에는 타겟 시스템에 개발된 프로그램을 로딩하고 프로그램의 연결 작업을 해 주는 링커/로더, 타겟에 로딩된 모듈들에서 정의/사용되는 심벌 정보를 관리하는 심벌 관리자, 타겟 시스템의 메모리를 관리하는 메모리 관리자가 포함되어 있다[8]. 호스트 시스템의 개발 환경에는 타겟 메모리와 심벌에 대한 정보가 저장되며 이러한 정보는 개발 환경의 도구들에 제공된다.

응용 프로그램 개발 과정은 [그림 2]와 같다. 프로그래머는 호스트 시스템에서 편집기를 이용하여 프로그램을 작성하고 크로스-컴파일러를 이용하여 컴파일 작업을 수행한다. 컴파일 된 모듈은 다른 모듈들과 연결되어 타겟 시스템으로 다운로드 되고 타겟 시스템에서 실행되게 된다. 프로그램의 수정이 필요한 경우 프로그램을 언로드한 후



[그림 2] 응용 프로그램 개발 과정

호스트 시스템에서 수정하고 다시 컴파일 하여 다운로드 한다. 이러한 과정은 프로그램 개발이 완료될 때까지 반복된다.

4. 점진적 링킹을 위한 심벌 관리

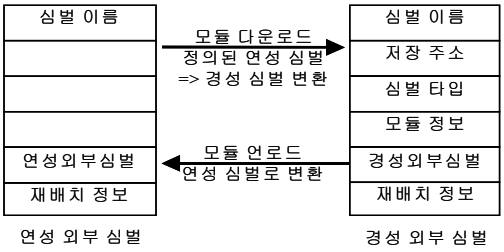
일반적으로 응용 프로그램은 여러 개의 파일로 이루어진다. 여러 개의 목적 파일은 외부 심벌의 참조로 서로 연관되어 있으며 프로그램 실행을 위해서는 외부 심벌의 연결 과정이 필요하다. 심벌 참조에 대한 연결을 위하여 통합 연결 방식(monolithic linking method)과 단위 모듈 연결 방식(individual linking method)을 사용할 수 있다. 통합 연결 방식은 모든 목적 파일을 하나로 연결하여 하나의 커다란 목적 파일을 타겟 시스템으로 전송하는 방법이다. 단위 모듈 연결 방식은 수정, 추가된 모듈에 관련된 연결 작업만을 수행하는 방식이다.

내장형 시스템용 응용 프로그램 개발자는 개발 초기에 작성된 일부 프로그램에 대한 테스트를 원하는 경우가 있다. 이를 위하여

불완전 연결 프로그램을 실행할 수 있는 방법이 필요하다. 이를 위해 외부 심벌을 두 가지로 구분하여 관리한다. 외부 심벌이 참조되는 시점에 정의가 이루어지지 않은 경우를 *연성 외부 심벌(Soft External Symbol)*이라 하고 심벌이 정의된 모듈이 존재하는 상태에서 참조가 이루어지는 경우 *경성 외부 심벌(Hard External Symbol)*이라 한다. 불완전 연결 프로그램의 실행은 연성 외부 심벌이 존재하는 상태에서의 프로그램 실행을 가능하게 해 주는 것이다. 연성 외부 심벌을 정의하는 모듈이 추가되면 타겟에 다운로드 되어 있는 모듈에서의 참조 부분과 연결하는 작업이 필요하다. 통합 연결 방식을 사용하게 되면 타겟 시스템에 다운로드 되어 있는 모든 모듈을 언로드하여 연결한 후 다시 다운로드 하는 과정이 필요하다. 이러한 연결 과정의 과부하를 방지하기 위하여 단위 모듈 연결 방식을 사용하여 점진적 링킹을 수행한다.

심벌이 참조되면 정의된 심벌 정보와 연결하기 위해 저장 주소, 타입과 같은 정보가 필요하다. 심벌 참조 부분과 정의된 심벌 정보를 연결하는 단계를 재배치 단계라 하며 모듈 연결 과정에서 외부 심벌에 대한 참조에 대해 처리하는 과정이다[2]. 재배치 단계는 경성 외부 심벌과 연성 외부 심벌에서 다르게 수행된다. 경성 외부 심벌의 경우 심벌이 정의된 상태에서 심벌에 대한 참조가 이루어지므로 심벌 참조를 중심으로 정의된 값을 연결한다. 이를 풀(pull)에 의한 링킹이라 한다. 연성 외부 심벌을 참조하는 모듈은 재배치에 필요한 심벌 정보가 존재하지 않으므로 참조 부분에 대한 재배치가 이루어지지 않은 상태에서 타겟 메모리로 다운로드 된다. 연성 외부 심벌의 정의가 포함된 모듈이 추가되면 심벌이 참조된 지점을 검색하여 재배치하게 되는데 이를 푸시(push)에 의한 링킹이라 한다.

연성 외부 심벌이 정의된 모듈이 다운로드 되면 저장 장소나 타입, 모듈 정보와 같은 심벌 정보가 추가되어 경성 외부 심벌로



[그림 3] 심벌 정보 변환

변환된다. 또한 경성 외부 심벌을 정의한 모듈이 언로드 되는 경우에는 경성 외부 심벌이 연성 외부 심벌로 변환된다. 이러한 심벌 정보의 변환 과정은 [그림 3]과 같다. 재배치 정보는 외부 심벌에 대한 참조가 있는 위치에서 저장된다. 하나의 심벌 정보는 여러 개의 재배치 정보를 가지며 리스트 형태로 저장된다. 연성 외부 심벌에서 경성 외부 심벌로 변환된 경우 푸시에 의한 링킹을 위하여 재배치 정보를 이용해야 하며 하나의 참조에 대해 하나의 재배치 정보를 사용한다. 재배치 정보는 재배치가 수행되어야 할 주소를 나타내는 섹션 주소와 옵셋 주소, 재배치 타입, 보정 값으로 이루어진다.

모듈의 적재 과정에서 재배치 대상인 외부 심벌이 연성 외부 심벌인 경우 재배치를 수행할 수 없으므로 재배치 정보만을 심벌 정보에 추가한다. 경성 외부 심벌을 참조하는 모듈이 다운로드 될 때는 심벌 참조를 중심으로 관련 경성 외부 심벌을 심벌 테이블에서 검색하여 재배치를 수행한다. 심벌 검색은 심벌 이름과 경성 외부 심벌이라는 분류 값을 이용하여 이루어진다. 연성 외부 심벌이 경성 외부 심벌로 변환되면 재배치 정보와 심벌 정보를 이용하여 재배치 주소 부분을 재배치된 값으로 수정한다. 목적 모듈이 언로드되면 참조되었던 심벌에 대한 재배치 정보를 삭제한다.

이러한 점진적 링킹 과정을 통해 응용 프로그램 개발 과정에서 통합 모듈 연결 방식을 사용하거나 심벌이 정의된 모듈을 참조하는 모듈보다 먼저 다운로드 해야 하는 불편을 없앨 수 있다.

5. 예외 처리

불완전 연결 프로그램을 실행할 수 있도록 하는 이유는 일반적으로 내장형 시스템용 응용 프로그램 개발자들이 시스템에 대해 잘 알고 있고 프로그램 개발 과정에 융통성을 줄 수 있기 때문이다. 내장형 시스템을 위한 응용 프로그램 개발자가 시스템과 그 시스템 상의 응용 프로그램의 특성을 파악하고 있는 경우 정상적인 실행 과정에서 수행될 부분을 알 수 있다. 일부 프로그램 모듈에 대한 테스트 과정에서 실행될 가능성이 없는 부분에 대해서는 연성 외부 심벌로 남겨두는 것이 문제가 되지 않는다. 그러나 연성 외부 심벌이 존재하는 상태에서 프로그램이 실행되면 프로그램 개발자의 의도와 다르게 연성 외부 심벌이 참조되는 경우 부적절한 메모리 접근으로 인한 예외 상황이 발생할 수 있다. 프로그램 실행 단계에서 연성 외부 심벌에 대한 참조가 이루어지면 적절한 정보로 재배치 되어있지 않은 상태이므로 메모리의 어떤 영역을 접근할지 예측할 수 없다. 이는 프로그램을 비정상적으로 종료시키며 시스템 영역 접근에 의하여 시스템이 다운될 수도 있다. 이러한 문제가 발생하면 응용 프로그램 개발 환경에 대한 신뢰성이 떨어지게 된다. 이러한 경우를 대비하여 예외 처리 기능을 제공한다.

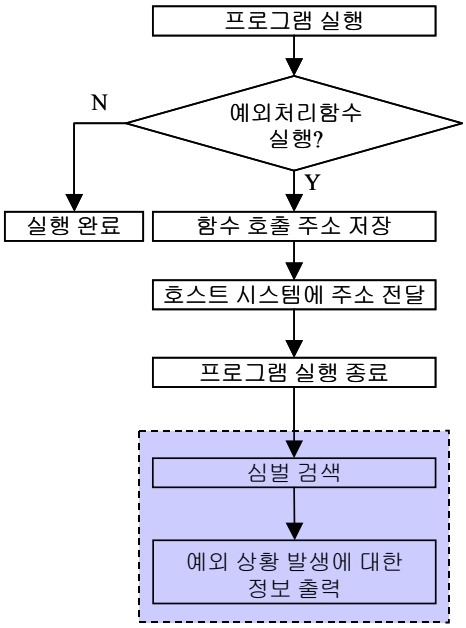
5.1 예외 처리 방법

예외 처리는 불완전 연결 프로그램을 실행하는 과정에서 연성 외부 심벌에 의해 발생할 수 있는 오류 문제를 해결하기 위하여 수행된다. 오류를 유발한 연성 외부 심벌 참조에 대한 정보를 제공하고 프로그램을 안정적으로 종료하는 기능을 수행한다. 이를 위해 프로그램을 안정적으로 종료하기 위한 내부 함수와 예외 상황 발생 위치를 저장하고 프로그램을 종료하기 위한 내부 함수로 컨트롤을 반환하는 역할을 담당하는 예외 처리 함수를 사용한다.

연성 외부 심벌에 대한 참조는 데이터를 접근하는 경우와 함수 호출 관련된 경우로

분류해볼 수 있다. 연성 외부 심벌이 데이터 접근에 관련된 경우 사용자 영역의 메모리를 할당하여 그 영역을 접근하도록 재배치한다. 이러한 경우 프로그램 개발자가 의도하지 않은 결과를 유발할 수 있으므로 미리 연성 외부 심벌의 존재에 대해 경고하도록 한다. 타겟 시스템의 프로그램 실행 단계에서 연성 외부 심벌의 리스트를 출력함으로써 사용자에게 동작의 오류 발생 가능성을 경고할 수 있다. 연성 외부 심벌을 참조하는 함수 호출의 경우 예외 처리 함수로 재배치한다. 이에 의해 실행 시간에 함수 호출 관련 연성 외부 심벌이 참조되면 예외 처리 함수가 실행되게 된다.

예외 상황에 의해 타겟 시스템에서의 프로그램 실행이 종료되면 예외 처리 함수에서 저장한 주소를 이용하여 예외 상황에 대한 정보를 검색한다. 연성 외부 심벌 참조에 의하여 예외 상황이 발생하는 경우 처리하는 알고리즘은 [그림 4]와 같다. 예외 처리 함수가 실행되면 함수 호출 관련 연성 외부 심벌이 참조된 것이므로 함수가 호출된 주소를 호스트 시스템에 전달하고 프로그램 실행을 종료한다. 예외 처리 함수는 함수 호출 관련 연성 외부 심벌 참조 지점에 재배치된 것이므로 예외 처리 함수를 호출한 주소를 저장하면 연성 외부 심벌이 참조된 주소를 알 수 있다. 심벌 검색과 예외 상황에 대한 정보 출력 작업은 호스트 시스템에서 이루어진다. 호스트 시스템에서는 전달된 오류 발생 주소와 심벌 테이블을 이용하여 예외 상황 발생에 대한 정보를 사용자에게 보고한다.



[그림 4] 예외 처리 알고리즘

예외 처리 함수에서 프로그램 실행을 안정적으로 종료하기 위한 방법이 필요하다. 프로그램의 안정적인 종료를 위하여 타겟 시스템에서의 실행을 전담하는 함수를 사용하였다. 타겟 시스템에 다운로드 된 프로그램을 실행하기 위해서는 실행할 함수 이름을 매개변수로 하여 동일한 내부 함수를 호출한다. 이 내부 함수는 실행 환경을 저장한 후 요청된 프로그램을 호출한다. 프로그램 실행 과정에서 예외 상황이 발생하면 예외 처리 함수에서 실행 환경을 복구하여 이 내부 함수로 돌아가며 프로그램을 정상적으로 종료하게 된다.

5.2 예외 처리를 위한 심벌 관리

연성 외부 심벌에 의한 예외 상황을 처리하는 과정에서 심벌 관리자는 두 가지 기능을

담당한다. 목적 모듈의 다운로드/언로드에 의한 심벌 변환 과정에서 재배치를 수행하기 위한 정보와 사용자에게 예외 상황에 대해 보고하기 위해 필요한 정보를 제공한다.

목적 모듈이 타겟 메모리에 다운로드 되면 해당 모듈에서 정의된 심벌이 경성 외부 심벌로 저장된다. 정의된 심벌이 심벌 테이블에 연성 외부 심벌로 존재하는 경우 경성 외부 심벌로 변환된다. 연성 외부 심벌은 임의의 사용자 메모리 영역 또는 예외 처리 함수로 재배치되어 있다. 경성 외부 심벌로 변환되면 정의된 심벌 정보를 이용하여 다시 재배치가 이루어져야 하므로 이를 위해 재배치 정보를 유지한다. 목적 모듈이 언로드 되면 해당 모듈에서 정의된 심벌들은 심벌 테이블에서 삭제되거나 연성 외부 심벌로 변환된다. 심벌을 참조하는 모듈이 타겟 메모리에 존재하는 경우 연성 외부 심벌로 변환되며 재배치를 다시 수행해야 한다. 데이터 접근의 경우 임의의 사용자 메모리 영역으로, 함수 호출의 경우 예외 처리 함수로 재배치하지 않으면 실행 시간에 예외 상황이 발생하거나 언로드 된 이전 모듈로 연결될 수 있다. 이런 경우 의도하지 않은 결과를 가져올 수 있으므로 예외 처리를 위한 재배치를 수행한다.

예외 상황이 발생했을 경우 예외 처리 함수는 자신을 호출한 주소를 호스트 시스템에 반환하게 된다. 이 주소는 함수 호출 관련 연성 외부 심벌이 참조된 주소이므로 심벌 테이블을 이용하여 예외 상황을 유발한 함수 호출 정보를 검색할 수 있다. 예를 들어 함수 func1()에서 func2()를 호출하는 지점에서 예외 상황이 발생했다면 func1은 경성 외부 심벌에서, func2는 연성 외부 심벌에서 검색한다. func1은 수행 중인 함수이므로 텍스트 타입 경성 외부 심벌이며 func2는 예외 처리 함수가 연결되어 있는 것으로 보아 연성 외부 심벌이다. 예외 상황 발생 주소는 함수 func1()의 영역에 포함되어 있으므로 심벌 테이블에서 경성 외부 심벌 중 예외 상황 발생 주소를 포함할 수 있는 텍스트 심벌을 검색하면 알 수 있다. 또한 오류 발생 주소를 연성 외부 심벌의 재배치 정보 중 재배치 주소와 비교하여 검색하면 func2()를 찾을 수 있다. 이와 같이 예외 발생 주소와 심벌 정보를 이용하여 사용자에게 예외 상황에 대한 정보를 보고한다.

6. 결론

프로그램 개발자는 응용 프로그램 작성을 위하여 편집, 컴파일, 실행 과정을 반복하여 수행한다. 이러한 과정에서 연성 외부 심벌의 존재를 허용하여 불완전 연결 프로그램의 실행을 가능하게 하였다. 이에 의해 개발 초기 단계의 테스트를 가능하게 하였고 내장형 응용 프로그램 개발 환경에서의 개발 과정에서 불필요한 프로그램 전송을 방지할 수 있었다.

불완전 연결 프로그램의 실행은 프로그래머에게 개발의 편의를 제공하지만 실행 시간의 오류를 발생시킬 수 있는 위험을 가지고

있다. 재배치가 완전히 이루어지지 않은 프로그램을 실행하는 과정에서 발생할 수 있는 부적절한 메모리 접근은 의도하지 않은 결과를 가져오거나 시스템을 다운시켜 시스템의 신뢰성을 저하시킬 수 있다. 불완전 연결 프로그램 실행 과정에서 발생하는 부적절한 메모리 접근 문제를 해결하기 위하여 예외 처리 방법을 도입하였다. 예외 처리 함수는 예외 상황이 발생하면 프로그램 실행을 정상적으로 종료시키고 예외 상황에 대한 정보를 사용자에게 보고한다.

최근 내장형 시스템과 개발 환경에 대한 연구가 활발하게 이루어지고 있다. 응용 프로그램 개발자에게 효율성과 편의성을 제공하기 위하여 불완전 연결 프로그램의 실행을 가능하게 하는 방법을 구현하였다. 또한 시스템의 신뢰성을 향상시키기 위하여 예외 처리 기법을 도입하였다.

현재의 예외 처리 기법은 시스템 오류 가능성이 있는 함수 호출 관련 연성 외부 심벌에 대한 처리에 치중하였다. 향후 데이터 접근 관련 심벌에 대한 다양한 예외 처리 방법을 연구해 나갈 수 있을 것으로 기대한다. 또한 예외 상황이 발생했을 때 사용자가 프로그램 종료 여부에 대해 선택하거나 데이터 값을 임의로 입력할 수 있게 하는 방향으로 발전시켜 나갈 수 있을 것이다.

참고문헌

[1] 우덕균, 한경숙, 표창우, 김흥남, "내장형 시스템을 위한 점진적이고 목표 재설정 가능한 링커", 정보과학회논문지 :컴퓨팅의 실제, 2001. 4

[2] John R, Levine, "Linkers and Loaders" Morgan Kaufmann Publishers, 2000

[3] WindRiver Systems, "Tornado user's guide", http://www.wrs.com

[4] WindRiver Systems, "Tornado", http://www.wrs.com

[5] David E. Simon, "An Embedded Software Primer", pp.261-281, Addison-Wesley, 1999

[6] R. W. Quong, M. A. Linton, "Linking Programs Incrementally", ACM Transactions on Programming Languages and Systems, Vol.13, No.1, pp.1-20, January 1991

[7] J. Xu, A. Romanovsky, B. Randell, "Concurrent Exception Handling and Resolution in Distributed Object Systems", IEEE Transactions on Parallel and Distributed Systems, 11(10), pp.19-32, October 2000

[8] 김흥남, "사용자 개발 도구", 정보가전용 실시간 OS 컨퍼런스 자료집, pp.178-196, 1999. 11“

한 경 숙

1993년 홍익대학교 컴퓨터공학과(학사)
1995년 홍익대학교 전자계산학과(석사)
2002년 홍익대학교 전자계산학과(박사)
2000년 5월 ~ 현재 (주)참좋은인터넷 재직

관심분야는 최적화 컴파일러, 내장형 시스템을 위한 응용 프로그램 개발 환경

표 창 우

1980년 서울대학교 전자공학과(학사)
1982년 서울대학교 컴퓨터공학과(석사)
1989년 Univ. of Illinois at Urbana Champaign 전산학과(박사)
1990년 1월 ~ 1991년 3월 Research Fellow. US Army Construction Engineering Research.
1991 4월 ~ 현재 홍익대학교 컴퓨터공학과 부교수
관심분야는 최적화 컴파일러, 내장형 시스템을 위한 응용 프로그램 개발 환경