(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.
G06F 9/00 (2006.01)

(45) 공고일자 2007년03월02일
(11) 등록번호 10-0640314
(24) 등록일자 2006년10월24일

(21) 출원번호 10-1998-0041792
(22) 출원일자 1998년10월07일
심사청구일자 2003년06월30일

(65) 공개번호 10-1999-0036883
(43) 공개일자 1999년05월25일

(30) 우선권주장 08/944,335 1997년10월06일 미국(US)

(73) 특허권자
썬 마이크로시스템즈, 인코포레이티드
미국 캘리포니아 95054 산타클라라 네트워크 써클 4150

(72) 발명자
배크 라스
미국, 캘리포니아 94303, 팔로 알토, 3782 코니아 웨이

글리셈머 로버트
미국, 캘리포니아 94025, 멘로 파크, #에프 960 오크 레인

휄즐레 어스
미국, 캘리포니아 93117, 골레타, #105, 7220 다벤포트 로드

(74) 대리인
강석용
강명구

심사관 : 성경아

전체 청구항 수 : 총 58 항

(54) 혼합된 실행 스택 및 예외처리의 구현방법및 그 장치

(57) 요약

다수의 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 한 실행 스택을 구현하기 위한 시스템 및 방법이 제공된다. 여러 각기 다른 프로그래밍 언어로 기록된 기능에 대한 프레임들이 같은 실행 스택상에서 사이에 끼워질 수 있다. 상기 실행 스택상의 데이터 블럭은 이전의 프레임으로의 프레임 포인터 및 스택 포인터를 저장하므로써 프레임을 거치지 않고 실행 스택을 가로지르도록 사용될 수 있다. 또한 필요하다면 각기 다른 프로그래밍 언어로 기록된 실행 스택상의 프레임을 통해 변환을 시킴으로써 예외가 전파될 수 있기도 하다.

대표도

도 4

특허청구의 범위

청구항 1.

컴퓨터 시스템에서 다수의 프로그래밍 언어로 기록된 함수들에 대한 프레임들을 저장하는 실행 스택을 구현하는 방법으로서, 상기 방법은,

- 제 1 프로그래밍 언어로 기록된 제 1 함수를 위해 상기 실행 스택에 제 1 프레임(461)을 저장하고,

- 제 2 프로그래밍 언어로 기록된 제 2 함수를 상기 제 1 함수가 호출함에 따라, 제 2 함수에 대한 제 2 프레임(455) 이전에 상기 실행 스택에 데이터 블록(457)을 저장하며, 이때, 상기 데이터 블록(457)은 제 2 프로그래밍 언어로 기록된 이전의 함수를 위한 상기 실행 스택에서의 이전 프레임(459)에 대한 한개 이상의 포인터를 포함하는

단계들을 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 2.

제 1 항에 있어서, 상기 한개 이상의 포인터는 스택 포인터(LAST_JAVA_SP)와 프레임 포인터(LAST_JAVA_FP)를 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 3.

제 1 항 또는 2 항에 있어서,

- 상기 제 1 함수가 상기 제 2 함수를 호출함에 따라, 상기 제 2 함수에 의해 호출될 수 있는 제 2 프로그래밍 언어와는 다른 프로그래밍 언어들로 기록된 함수들을 위한 자원을 할당하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 4.

제 3 항에 있어서,

- 제 2 함수를 빠져나오면, 제 2 프로그래밍 언어와는 다른 프로그래밍 언어들로 기록된 함수들을 위한 자원을 할당해제하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 5.

제 1 항 또는 2 항에 있어서,

- 상기 제 2 함수를 위한 예외 처리기(exception handler)에 의해 처리되지 않은, 제 2 함수의 실행 중 발생하였던 예외를 캐치(catch)하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 6.

제 5 항에 있어서,

- 상기 예외를 처리하도록 상기 데이터 블록에 대한 예외 처리기를 식별하고, 그리고 상기 식별된 예외 처리기로 점프하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 7.

제 6 항에 있어서, 상기 식별된 예외 처리기는 로컬 기억장치에 상기 예외를 저장하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 8.

제 7 항에 있어서, 상기 로컬 기억장치는 상기 제 1 함수와 상기 제 2 함수를 실행하고 있는 현재 쓰레드에 관련된 기억 장치인 것을 특징으로 하는 실행 스택 구현 방법.

청구항 9.

제 7 항에 있어서,

- 상기 제 1 함수로 복귀함에 따라, 예외가 미정(pending)인 지를 결정하기 위해 상기 로컬 기억장치를 조사하고, 그리고 상기 예외가 미정일 경우 저장된 예외를 발생(throw)시키는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 10.

제 9 항에 있어서,

- 상기 저장된 예외를 상기 제 1 프로그래밍 언어를 위한 포맷으로 변환하는 단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 11.

제 1 항 또는 2 항에 있어서, 상기 제 2 프로그래밍 언어가 자바 프로그래밍 언어인 것을 특징으로 하는 실행 스택 구현 방법.

청구항 12.

다수의 프로그래밍 언어로 기록된 함수들에 대한 프레임들을 저장하는 실행 스택을 구현하는 컴퓨터 코드를 포함하는 컴퓨터-판독형 매체로서, 상기 컴퓨터-판독형 매체는,

- 제 1 프로그래밍 언어로 기록된 제 1 함수를 위해 상기 실행 스택에 제 1 프레임(461)을 저장하는 컴퓨터 코드, 그리고

- 제 2 프로그래밍 언어로 기록된 제 2 함수를 상기 제 1 함수가 호출함에 따라, 제 2 함수에 대한 제 2 프레임(455) 이전에 상기 실행 스택에 데이터 블록(457)을 저장하는 컴퓨터 코드로서, 이때, 상기 데이터 블록(457)은 제 2 프로그래밍 언어로 기록된 이전의 함수를 위한 상기 실행 스택에서의 이전 프레임(459)에 대한 한개 이상의 포인터를 포함하는 바의 컴퓨터 코드

를 저장한느 것을 특징으로 하는 컴퓨터-판독형 매체.

청구항 13.

제 12 항에 있어서, 상기 컴퓨터-판독형 매체는 CD-ROM, 플라피 디스크, 테이프, 플래시 메모리, 시스템 메모리, 그리고, 하드 드라이브 중 한가지인 것을 특징으로 하는 컴퓨터-판독형 매체.

청구항 14.

다수의 프로그래밍 언어로 기록된 함수들에 대한 프레임들을 저장하는 실행 스택을 구현하는 컴퓨터 시스템으로서, 상기 컴퓨터 시스템은,

- 프로세서, 그리고

- 상기 프로세서에 연결되어 상기 실행 스택을 저장하는 메모리

를 포함하며,

상기 프로세서는 제 1 프로그래밍 언어로 기록된 제 1 함수를 위해 상기 실행 스택에 제 1 프레임(461)을 저장시키도록 동작하고, 제 2 프로그래밍 언어로 기록된 제 2 함수를 상기 제 1 함수가 호출함에 따라, 상기 프로세서는 제 2 함수에 대한 제 2 프레임(455) 이전에 상기 실행 스택에 데이터 블록(457)을 저장시키도록 동작하며, 이때, 상기 데이터 블록(457)은 제 2 프로그래밍 언어로 기록된 이전의 함수를 위한 상기 실행 스택 상의 이전의 프레임(459)에 대한 한개 이상의 포인터를 포함하는 것을 특징으로 하는 컴퓨터 시스템.

청구항 15.

컴퓨터 시스템에서 다수의 프로그래밍 언어로 기록된 함수들에 대한 프레임들을 저장하는 실행 스택을 구현하는 방법으로서, 상기 방법은,

- 제 1 프로그래밍 언어로 기록된 제 1 함수를 위해 상기 실행 스택에 제 1 프레임을 저장하고,

- 제 2 프로그래밍 언어로 기록된 제 2 함수를 상기 제 1 함수가 호출함에 따라, 상기 실행 스택 상의 제 1 프레임에 대한 한개 이상의 포인터를 로컬 기억장치에 저장하고, 그리고 상기 제 2 함수를 위해 상기 실행 스택에 제 2 프레임을 저장하는

단계를 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 16.

제 15 항에 있어서, 상기 한개 이상의 포인터는 스택 포인터(LAST_JAVA_SP)와 프레임 포인터(LAST_JAVA-FP)를 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 17.

제 15 항 또는 16 항에 있어서, 상기 로컬 기억장치는 제 1 함수와 제 2 함수를 실행하고 있는 현재 쓰레드에 관련된 기억장치인 것을 특징으로 하는 실행 스택 구현 방법.

청구항 18.

제 15 항에 있어서,

- 제 2 함수를 빠져나올 때, 상기 로컬 기억 장치에 저장된 한개 이상의 포인터를 삭제하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 19.

제 15 항에 있어서,

- 상기 제 2 함수를 위한 예외 처리기(exception handler)에 의해 처리되지 않은 제 2 함수의 실행 중 발생하였던 예외를 캐치(catch)하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 20.

제 19 항에 있어서,

- 상기 예외가 상기 제 1 프로그래밍 언어에 대하여 적합한 지를 결정하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 21.

제 19 항에 있어서,

- 상기 예외를 상기 로컬 기억장치에 저장하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 22.

제 19 항에 있어서,

- 상기 실행 스택 상의 리턴 주소를 예외 포워드기(exception forwarder)의주소로 패치(patch)하며, 이때, 상기 예외 포워드기는 상기 예외를 처리하기 위해 상기 제 1 함수에 대한 예외 처리기를 식별하고, 그리고 상기 식별된 예외 처리기로 점프하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 23.

제 22 항에 있어서, 상기 예외 포워드기가 상기 예외를 제 1 프로그래밍 언어에 대한 포맷으로 변환하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 24.

제 15 항에 있어서,

- 상기 제 1 프로그래밍 언어로 기록된 제 3 함수를 상기 제 2 함수가 호출함에 따라, 상기 제 3 함수에 대한 제 3 프레임 이전에 상기 실행 스택에 데이터 블록을 저장하고, 이때, 상기 데이터 블록은 로컬 기억장치에 저장된 상기 제 1 프레임에 대한 한개 이상의 포인터를 포함하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 25.

제 24 항에 있어서,

- 상기 제 3 함수를 상기 제 2 함수가 호출함에 따라, 상기 제 3 함수에 의해 호출될 수 있는 제 1 프로그래밍 언어와는 다른 프로그래밍 언어로 기록된 함수들에 대한 자원들을 할당하고, 그리고

- 상기 제 3 함수를 빠져나옴에 따라, 상기 제 1 프로그래밍 언어와는 다른 프로그래밍 언어로 기록된 함수들에 대한 자원들을 할당해제하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 26.

제 24 항에 있어서,

- 상기 제 3 함수에 대한 예외 처리기에 의해 처리되지 않은 제 3 함수의 실행 중 발생된 예외를 캐치(catch)하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 27.

다수의 프로그래밍 언어로 기록된 함수들에 대한 프레임들을 저장하는 실행 스택을 구현하는 컴퓨터 코드를 포함하는 컴퓨터-판독형 매체로서, 상기 컴퓨터-판독형 매체는,

- 제 1 프로그래밍 언어로 기록된 제 1 함수를 위해 상기 실행 스택에 제 1 프레임을 저장하는 컴퓨터 코드, 그리고

- 제 2 프로그래밍 언어로 기록된 제 2 함수를 상기 제 1 함수가 호출함에 따라, 상기 실행 스택 상의 제 1 프레임에 대한 한개 이상의 포인터를 로컬 기억장치에 저장하고, 그리고 상기 제 2 함수를 위해 상기 실행 스택에 제 2 프레임을 저장하는 컴퓨터 코드

를 저장하는 것을 특징으로 하는 컴퓨터-판독형 매체.

청구항 28.

실행 스택을 구현하기 위한 데이터 구조를 저장하기 위한 컴퓨터-판독형 매체로서, 상기 컴퓨터-판독형 매체는,

- 제 1 프로그래밍 언어로 기록된 제 1 함수를 위해 상기 실행 스택 상에 위치하는 제 1 프레임,

- 제 2 프로그래밍 언어로 기록된 제 2 함수를 위해 상기 제 1 프레임 위의 실행 스택에 위치하는 제 2 프레임, 그리고

- 상기 제 2 프레임 위의 실행 스택에 위치하는 데이터 블록으로서, 상기 데이터 블록은 상기 실행 스택 상의 제 1 프레임에 대한 한개 이상의 포인터를 포함하는 바의 상기 데이터 블록

을 저장하는 것을 특징으로 하는 컴퓨터-판독형 매체.

청구항 29.

제 3 항에 있어서,

- 상기 제 2 함수를 위한 예외 처리기(exception handler)에 의해 처리되지 않은 제 2 함수의 실행 중 발생하였던 예외를 캐치하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 30.

제 4 항에 있어서,

- 상기 제 2 함수를 위한 예외 처리기(exception handler)에 의해 처리되지 않은 제 2 함수의 실행 중 발생하였던 예오를 캐치하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 31.

컴퓨터 시스템에서 다수의 프로그래밍 언어로 기록된 함수들에 대한 자원들을 할당하는 방법으로서, 상기 방법은,

- 제 1 프로그래밍 언어로 기록된 제 1 함수를 위해 상기 실행 스택에 제 1 프레임을 저장하고,

- 상기 제 1 함수를 이용하여 제 2 함수를 호출하며, 그리고

- 상기 제 2 함수에 의해 호출될 수 있는 제 2 프로그래밍 언어와는 다른 프로그래밍 언어로 기록된 함수들에 대해 자원들을 할당하는

단계들을 포함하는 것을 특징으로 하는 자원 할당 방법.

청구항 32.

제 31 항에 있어서,

- 제 2 함수를 빠져나옴에 따라, 상기 제 2 프로그래밍 언어와는 다른 프로그래밍 언어로 기록된 함수들에 대한 자원들을 할당 해제하는

단계를 추가로 포함하는 것을 특징으로 하는 자원 할당 방법.

청구항 33.

제 31 항에 있어서, 상기 제 1 함수를 이용하여 제 2 함수를 호출하는 상기 단계는 상기 제 2 함수를 실행하는 단계를 포함하고, 상기 방법은,

- 상기 제 2 함수에 대한 예외 처리기에 의해 처리되지 않은 제 2 함수의 실행 중 발생된 예외를 캐치(catch)하는

단계를 추가로 포함하는 것을 특징으로 하는 자원 할당 방법.

청구항 34.

제 33 항에 있어서,

- 상기 예외를 처리하기 위한 데이터 블록에 대한 예외 처리기를 식별하고, 그리고,

- 상기 제 2 함수로부터 상기 식별된 예외 처리기로 점프하는

단계를 추가로 포함하는 것을 특징으로 하는 자원 할당 방법.

청구항 35.

제 34 항에 있어서, 상기 식별된 예외 처리기는, 상기 제 1 함수를 이용하여 제 2 함수를 호출하는 스레드에 관련된 로컬 기억장치에 상기 예외를 저장하는 것을 특징으로 하는 자원 할당 방법.

청구항 36.

제 35 항에 있어서,

- 상기 제 2 함수로부터 상기 제 1 함수로 복귀하고,

- 상기 예외가 미정(pending)인 지를 결정하기 위해 상기 로컬 기억장치를 조사하며, 그리고

- 상기 예외가 미정(pending)이라고 결정되면 상기 저장된 예외를 발생(throw)시키는

단계를 추가로 포함하는 것을 특징으로 하는 자원 할당 방법.

청구항 37.

컴퓨터 시스템에서 다수의 프로그래밍 언어로 기록된 함수들에 대한 예외들을 처리하는 방법으로서, 상기 방법은,

- 제 1 프로그래밍 언어로 기록된 제 1 함수를 위해 상기 실행 스택에 제 1 프레임을 저장하고,

- 상기 제 1 함수를 이용하여 제 2 함수를 호출하며, 그리고

- 상기 제 2 함수에 대한 예외 처리기에 의해 처리되지 않은 제 2 함수의 실행 중 발생된 예외를 캐치(catch)하는

단계를 포함하는 것을 특징으로 하는 예외 처리 방법.

청구항 38.

제 37 항에 있어서,

- 상기 예외를 처리하기 위한 데이터 블록에 대한 예외 처리기를 식별하고, 그리고, 상기 식별된 예외 처리기로 점프하는

단계를 추가로 포함하는 것을 특징으로 하는 예외 처리 방법.

청구항 39.

제 38 항에 있어서, 상기 식별된 예외 처리기는 로컬 기억장치에 상기 예외를 저장하는 것을 특징으로 하는 예외 처리 방법.

청구항 40.

제 39 항에 있어서, 상기 로컬 기억장치는, 제 1 함수 및 제 2 함수가 실행되고 있는 현재의 쓰레드에 관련된 기억장치인 것을 특징으로 하는 예외 처리 방법.

청구항 41.

제 39 항에 있어서,

- 예외가 미정(pending)인 지를 결정하기 위해 상기 로컬 기억장치를 조사하고,

- 상기 예외가 미정(pending)일 경우 상기 저장된 예외를 발생(throw)시키는

단계를 추가로 포함하는 것을 특징으로 하는 예외 처리 방법.

청구항 42.

제 41 항에 있어서,

- 상기 저장된 예외를 상기 제 1 프로그래밍 언어에 대한 포맷으로 변환하는

단계를 추가로 포함하는 것을 특징으로 하는 예외 처리 방법.

청구항 43.

제 37 항에 있어서, 제 1 함수를 이용하여 제 2 함수를 호출하는 상기 단계는, 상기 실행 스택에 데이터 블록을 저장하는 단계를 포함하고,

상기 데이터 블록은, 제 1 프로그래밍 언어로 기록되지 않은 제 3 함수를 위한 상기 실행 스택의 또다른 프레임에 대한 참조(reference)를 포함하는 것을 특징으로 하는 예외 처리 방법.

청구항 44.

컴퓨터 시스템에서 다수의 프로그래밍 언어들로 기록된 함수들에 대한 프레임들을 저장하는 실행 스택을 구현하는 방법으로서, 상기 방법은,

- 제 1 프로그래밍 언어로 기록된 제 1 함수에 대해 상기 실행 스택에 제 1 프레임을 저장하고,

- 제 2 프로그래밍 언어로 기록된 제 2 함수를 위해 상기 실행 스택에 제 2 프레임을 저장하며, 그리고

- 상기 제 2 함수로부터 제 3 함수를 호출하고, 이때, 상기 제 3 함수는 제 1 프로그래밍 언어로 기록되는

단계들을 포함하고, 이때, 상기 제 2 함수로부터 제 3 함수를 호출하는 상기 단계는 제 1 프레임에 대한 한개 이상의 참조(reference)를 포함하는 데이터 블록을 상기 실행 스택에 저장하는 단계를 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 45.

제 44 항에 있어서,

- 상기 실행 스택에 데이터 블록을 저장한 후, 상기 제 3 함수에 대한 제 3 프레임을 상기 실행 스택에 저장하는

단계를 추가로 포함하는 것을 특징으로 하는 실행 스택 구현 방법.

청구항 46.

제 44 항에 있어서, 상기 제 1 프로그래밍 언어가 자바 프로그래밍 언어인 것을 특징으로 하는 실행 스택 구현 방법.

청구항 47.

제 46 항에 있어서, 상기 제 2 프로그래밍 언어는 C++, 파스칼, 포트란, 그리고 어셈블리 언어 중 한가지인 것을 특징으로 하는 실행 스택 구현 방법.

청구항 48.

연산 환경에서 다수의 프로그래밍 언어를 지원할 수 있는 혼합 실행 스택을 구현하는 방법으로서, 상기 방법은,

- 상기 혼합 실행 스택에 제 1 프레임을 저장하고, 이때, 상기 제 1 프레임은 제 1 프로그래밍 언어로 기록된 컴퓨터 프로그램 코드의 실행을 위한 프레임이며,

- 제 2 프로그래밍 언어로 기록된 컴퓨터 프로그램 코드의 실행을 위한 제 2 프레임이 상기 혼합 실행 스택에 저장될 지를 결정하며, 이때, 상기 제 1, 2 프로그래밍 언어는 서로 다른 프로그래밍 언어이며,

- 상기 제 2 프로그래밍 언어로 기록된 상기 컴퓨터 프로그램 코드의 실행을 위한 상기 제 2 프로그램이 상기 혼합 실행 스택에 저장될 것이라고 결정되었을 때, 상기 제 1 프레임에 대한 한개 이상의 참조(reference)를 저장하는

단계를 포함하는 것을 특징으로 하는 혼합 실행 스택 구현 방법.

청구항 49.

제 48 항에 있어서,

- 상기 혼합 실행 스택에 상기 제 2 프레임을 저장하는

단계를 추가로 포함하는 것을 특징으로 하는 혼합 실행 스택 구현 방법.

청구항 50.

제 48 항에 있어서,

- 상기 제 1 프로그래밍 언어에 관련된 제 3 프레임이 상기 혼합 실행 스택에 저장될 지를 결정하고, 그리고

- 상기 제 3 프레임이 상기 혼합 실행 스택에 저장된다고 결정될 때, 상기 실행 스택의 상기 제 1 프레임에 대한 상기 한개 이상의 참조를 저장하는

단계를 추가로 포함하는 것을 특징으로 하는 혼합 실행 스택 구현 방법.

청구항 51.

제 48 항에 있어서,

- 상기 제 1 프로그래밍 언어에 관련된 제 3 프레임이 상기 혼합 실행 스택에 저장될 지를 결정하고, 그리고

- 상기 제 3 프레임이 상기 혼합 실행 스택에 저장되기 전에 상기 혼합 실행 스택에 엔트리 프레임을 저장하는

단계를 추가로 포함하는 것을 특징으로 하는 혼합 실행 스택 구현 방법.

청구항 52.

제 48 항에 있어서, 상기 제 1 프레임에 대한 한개 이상의 참조를 저장하는 상기 단계는, 상기 한개 이상의 참조를 로컬 기억장치에 저장하는 단계를 포함하는 것을 특징으로 하는 혼합 실행 스택 구현 방법.

청구항 53.

제 48 항에 있어서, 상기 제 1 프레임에 대한 한개 이상의 참조를 저장하는 상기 단계는, 상기 한개 이상의 참조를 로컬 기억장치에 저장하고, 그후 상기 혼합 실행 스택에 상기 한개 이상의 참조를 저장하는 단계를 포함하는 것을 특징으로 하는 혼합 실행 스택 구현 방법.

청구항 54.

제 48 항에 있어서, 상기 제 1 프로그래밍 언어 또는 상기 제 2 프로그래밍 언어가 자바 프로그래밍 언어인 것을 특징으로 하는 혼합 실행 스택 구현 방법.

청구항 55.

연산 환경에서 다수의 프로그래밍 언어들을 지원할 수 있는 혼합 실행 스택을 이용할 수 있는 가상 머신을 포함하는 장치로서, 상기 가상 머신은,

- 상기 혼합 실행 스택에 제 1 프레임을 저장하고, 이때, 상기 제 1 프레임은 제 1 프로그래밍 언어로 기록된 컴퓨터 프로그램 코드의 실행을 위한 프레임이며,

- 제 2 프로그래밍 언어로 기록된 컴퓨터 프로그램 코드의 실행을 위한 제 2 프레임이 상기 혼합 실행 스택에 저장될 지를 결정하며, 이때, 상기 제 1, 2 프로그래밍 언어는 서로 다른 프로그래밍 언어이며,

- 상기 제 2 프로그래밍 언어로 기록된 상기 컴퓨터 프로그램 코드의 실행을 위한 상기 제 2 프로그램이 상기 혼합 실행 스택에 저장될 것이라고 결정되었을 때, 상기 제 1 프레임에 대한 한개 이상의 참조(reference)를 저장하는

동작들을 실행하는 것을 특징으로 하는 가상 머신을 포함하는 장치.

청구항 56.

제 55 항에 있어서, 상기 가상 머신은

- 상기 혼합 실행 스택에 상기 제 2 프레임을 저장하는

동작을 추가로 실행하는 것을 특징으로 하는 가상 머신을 포함하는 장치.

청구항 57.

제 55 항에 있어서, 상기 가상 머신은,

- 상기 제 1 프로그래밍 언어에 관련된 제 3 프레임이 상기 혼합 실행 스택에 저장되는 지를 결정하고, 그리고

- 상기 제 3 프레임이 상기 혼합 실행 스택에 저장된다고 결정되었을 때, 상기 실행 스택의 상기 제 1 프레임에 대한 상기 한개 이상의 참조를 저장하는

동작을 추가로 실행하는 것을 특징으로 하는 가상 머신을 포함하는 장치.

청구항 58.

제 55 항에 있어서, 상기 제 1 프로그래밍 언어나 제 2 프로그래밍 언어가 자바 프로그래밍 언어인 것을 특징으로 하는 가상 머신을 포함하는 장치.

명세서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 실행스택의 구현에 대한 것이다. 특히 본 발명은 가령 자바(Java) 가상머신을 위한 실행스택과 같은 다중 프로그래밍 언어로 기록될 수 있는 기능과 함께 사용하기 위한 실행스택을 구현함에 대한 것이다.

상기 자바 프로그래밍 언어는 팔로 알토, 캘리포니아의 선 마이크로시스템스에 의해 개발된 객체지향형 고수준 프로그래밍 언어이다. 그리고 상기 언어는 소형 개인용 컴퓨터 슈퍼 컴퓨터에 이르는 광범위한 컴퓨터에서 실행되기에 충분한 포터블로 만들어진다. 자바(및 다른 언어)로 기록한 컴퓨터 프로그램은 자바 가상 머신에 의해 실행하기 위한 가상 머신 지시로 컴파일될 수 있다. 일반적으로 자바 가상머신은 가상머신 지시를 해독하고 실행시키는 인터프리터이다.

자바 가상머신을 위한 상기 가상머신 지시는 바이트 코드이며 이들 바이트 코드는 하나 또는 둘이상의 바이트를 포함함을 의미한다. 상기 바이트 코드는 한 클래스의 방법을 위한 바이트 코드를 포함하는 클래스 파일이라 불리는 특정 파일 포맷으로 저장된다. 한 클래스의 방법에 대한 바이트 코드에 추가하여 상기 클래스 파일은 다른 보조 정보뿐 아니라 한 기호 테이블을 포함한다.

하나 또는 둘이상의 클래스 파일로 자바 바이트 코드로써 실시된 컴퓨터 프로그램은 플랫폼 인디펜던트이다. 상기 컴퓨터 프로그램은 자바 가상머신의 실시를 수행할 수 있는 어떠한 컴퓨터에서도 실행되고 수정되지 않을 수 있다. 상기 자바 가상머신은 자바 가상머신을 위한 컴퓨터 프로그램이 플랫폼 인디펜던트이도록 하는데 있어서 주요한 요소인 일반 컴퓨터의 소프트웨어 에뮬레이터이다.

상기 자바 가상머신은 한 소프트웨어 인터프리터로써 통상 실시된다. 종래의 인터프리터들은 실행중 한번에 한 지시씩 해석된 프로그램의 가상머신 지시를 해독하고 실행한다. 반면 컴파일러들은 해독이 실행중에 수행되지 않도록 실행이전에 소스 코드를 고유 머신지시로 해독시킨다. 종래의 인터프리터들이 지시가 있을때마다 반복해서 실행되기 전에 각 지시를 해독하기 때문에 해석된 프로그램의 실행은 대개 컴파일된 프로그램보다 매우 느린데 이는 컴파일된 프로그램의 고유 머신지시들이 해독을 필요로 하지 않고 고유 머신 또는 컴퓨터 시스템상에서 실행될 수 있기 때문이다.

대개 자바 가상머신은 자바 프로그래밍 언어가 아닌 프로그래밍 언어(가령 C++ 프로그래밍 언어)로 기록될 수 있을 것이다. 따라서 자바 프로그래밍 실행은 다중 프로그래밍 언어로 기록된 기능의 실행을 포함할 수 있다. 추가로 바이트 코드 자신들은 자바 프로그래밍 언어로 기록되지 않은 기능을 불러들일 수도 있다(가령 입출력용 시스템 기능). 따라서 실행하는 자바 프로그램은 다중 프로그래밍 언어로 기록되었던 기능의 실행을 포함하는 것이 일반적이다. 다중 프로그래밍 언어로 기록된 기능을 저장하는 한 실행 스택을 갖는 것이 바람직 할 것이다. 또한 필요한때에는 적절한 포맷으로의 예외 변환과 함께 이들 기능을 통해 예외가 전파될 수 있도록 하는 것이 유익할 것이다.

자바 가상머신은 자바 프로그램이 고유 방법을 동적으로 적재하고 실행할 수 있도록 하며 이때 한 고유 방법이 자바 프로그래밍 언어가 아닌 언어로 실시된다. 고유 방법들은 실행하기 위한 자원을 자주 필요로 하며 따라서 자원들이 고유 방법이 호출될때마다 할당되고 할당 해제될 수 있도록 된다. 그러나 자원할당 및 할당 해제는 매우 시간 소모적이며 계속적인 할당 및 자원의 할당 해제는 해석된 프로그램의 효율을 크게 떨어뜨리게 된다. 따라서 자원 할당 및 할당 해제에 대한 보다 효율적인 기술을 제공하는 것이 바람직하다.

따라서 각기 다른 프로그래밍 언어로 실행된 기능을 위한 기능시동 유지를 가능하게 하는 실행 스택을 실시하기 이한 기술이 필요하며 특히 이들 기능 및 상응하는 시동을 통해 예외가 전파될 수 있도록 하는 기술이 필요하다. 추가로 실행 속도가 증가될 수 있도록 자원이 할당되고 할당 해제되는 방법으로 더욱 효율적인 인터프리터를 제공할 필요가 있다.

발명이 이루고자 하는 기술적 과제

본 발명의 실시예는 여러 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 실행 스택을 실현하기 위한 혁신적인 시스템 및 방법을 제공한다. 여러 프로그래밍 언어로 기록된 기능에 대한 프레임은 같은 프로그래밍 언어로 기록된 기능에 대한 실행 스택에서 한 이전의 프레임으로의 포인터를 저장하는 엔트리 프레임(또는 데이터 블럭)을 사용하는 같은 실행 스택상에서 저장될 수 있다. 상기 엔트리 프레임은 상기 실행 스택이 다른 프로그래밍 언어로 기록된 기능에 대한 프레임을 "거치지 않고(around)" 횡단될 수 있도록 하며 예외가 가령 그 기능이 다른 프로그래밍 언어로 기록되는 때 그리고 그 예외의 포맷이 다른 때에도 적절한 예외 처리기에 의해 처리하기 위해 상기 실행 스택을 통해 전파될 수 있다. 또한 자원이 한 프로그래밍 언어(가령 자바 프로그래밍 언어)로 기록된 기능을 입력시키자 마자 할당될 수 있어서 상기 자원들이 다른 프로그래밍 언어로 된 뒤이은 기능(가령 고유)에서도 이용될 수 있도록 할 것이다. 상기 자원들은 호출기능이 종료되기만 하면 할당 해제될 수 있어서 자원 할당 및 할당 해제의 효율을 증가시키도록 한다. 본 발명의 여러 실시예가 하기에서 설명된다.

한 실시예에서 여러 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 한 실행 스택을 구현하기 위한 컴퓨터에 의해 구현된 방법이 제공된다. 첫 번째 프로그래밍 언어로 기록된 기능에 대한 첫 번째 프레임이 상기 실행 스택상에 저장될 수 있다. 상기 첫 번째 기능이 두 번째 프로그래밍 언어로 기록된 두 번째 기능을 호출하는때 데이터 블럭이 상기 두 번째 기능에 대한 두 번째 프레임 이전에 상기 실행 스택상에 저장될 수 있다. 상기 데이터 블럭은 첫 번째 프로그래밍 언어로 기록된 상기 실행 스택상에서 이전 프레임으로의 적어도 한 포인터를 포함한다. 바람직한 실시예에서 데이터 블럭은 상기 실행스택상에서 이전 프레임으로의 한 스택 포인터와 한 프레임 포인터를 저장하며 상기 스택포인터와 프레임 포인터는 로컬 기억장치(쓰레드 로컬 기억장치)에 저장되었다.

또다른 실시예에서 여러 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 한 실행 스택을 구현하기 위한 컴퓨터에 의해 구현된 방법이 제공된다. 첫 번째 프로그래밍 언어로 기록된 기능에 대한 첫 번째 프레임이 상기 실행스택상에 저장될 수 있다. 상기 첫 번째 기능이 두 번째 프로그래밍 언어로 기록된 두 번째 기능을 호출하는때 상기 실행스택상에서 첫 번째 프레임으로의 적어도 한 포인터가 로컬 기억장치(가령 쓰레드 로컬 기억장치)내에 저장될 수 있다. 상기 적어도 한 포인터는 스택 포인터 및 프레임 포인터일 수 있다. 다음에 상기 첫 번째 프로그래밍 언어로 기록된 세 번째 기능이 호출되는때 한 데이터 블럭이 상기 세 번째 기능에 대한 세 번째 프레임 이전에 상기 실행스택상에 저장될 수 있다. t 상기 데이터 블럭은 로컬 기억장치내에 저장되었던 상기 실행스택상의 첫 번째 프레임으로 적어도 하나의 포인터를 저장하여 상기 데이터 블럭이 상기 실행스택상에 두 번째 프레임을 지나지 않는 방법을 제공하도록 한다. 바람직한 실시예에서 상기 첫 번째 프로그래밍 언어는 자바 프로그래밍 언어이다.

또다른 실시예에서 한 실행스택을 구현하기 위한 컴퓨터 판독가능 매체에 의해 저장된 한 데이터 구조가 제공된다. 첫 번째 프레임은 첫 번째 프로그래밍 언어로 기록된 첫 번째 기능을 위해 상기 실행스택상에 저장된다. 두번째 프레임은 두번째 프로그래밍 언어로 기록된 두번째 기능을 위해 상기 실행 스택상에 저장된다. 상기 실행 스택의 두번째 프레임위 상기 실행 스택상에 한 데이터 블럭이 저장되며 이는 상기 실행 스택상의 첫번째 프레임으로의 적어도 한 포인터를 포함한다. 한 바람직한 실시예에서 상기 데이터 블럭은 첫번째 프레임으로의 스택 포인터 및 프레임 포인터를 저장한다.

하기에서는 첨부도면을 참조하여 본 발명을 상세히 설명한다.

발명의 구성

정의

기능 - 소프트웨어 루틴(서브 루틴, 절차, 멤버기능 및 방법으로 불리기도 한다).

프레임(또는 시동 프레임, 시동 기록) - 상기 기능의 실행을 위한 정보를 저장하기 위해 한 기능에 대하여 실행 스택상에 저장된 기록, 상기의 정보는 상태 변수, 로컬변수 및 피연산자 스택을 포함할 수 있다.

실행 스택 - 순차적인 호출순서로 기능에 대한 프레임을 저장하기 위해 프로그램 실행중에 사용된 스택. 한 기능(피호출자)이 호출되는때 상기 기능에 대한 프레임이 상기 실행스택상에서 푸쉬된다. 뒤이어 상기 기능이 종료된때 프레임은 스택으로부터 나오며 상기 실행 스택의 상위에 있는 프레임에 대한 기능(호출자)이 실행을 재개한다.

피연산자 스택 - 실행중 머신지시에 의해 사용하기 위해 피연산자를 저장하기 위해 사용된 스택.

외부 코드 - 특정 프로그래밍 언어(가령 자바 프로그래밍 언어)가 아닌 한 프로그래밍 언어로 기록된 컴퓨터 코드. 예를 들어 자바 가상머신이 C++ 프로그래밍 언어로 기록될 수 있으며 상기 자바 가상머신은 해석되고 있는 프로그램의 자바 코드를 참조로 하여 외부 코드인 것으로 생각될 수 있다. 외부 코드는 고유 방법을 포함한다.

고유 방법 - 자바 프로그래밍 언어가 아닌 한 프로그래밍 언어로 기록된 기능. 고유 방법은 이들이 동적으로 적재되고 실행되도록 하는 자바 프로그램에 의해 호출될 수 있다. 추가로 고유 방법들은 자바 프로그래밍 언어로 기록된 기능을 호출할 수 있다.

개관

다음의 설명에서는 본 발명이 자바 프로그램(가령 바이트 코드)을 실행하는 자바 가상머신을 위한 한 실행 스택을 구현하는 바람직한 실시예를 참조로 하여 설명될 것이다. 특히 이 예들은 자바 가상머신이 C++ 프로그래밍 언어로 기록됨을 설명한다. 그러나 본 발명은 특정 언어, 컴퓨터 아키텍쳐 또는 특정 구현으로 제한되지 않는다. 따라서 실시예의 설명은 설명을 위한 목적인 것이고 제한을 위한 것은 아니다.

도 1 은 본 발명 실시예의 소프트웨어를 실시하기 위해 사용될 수 있는 컴퓨터 시스템의 한 예를 도시한 것이다. 도 1 은 표시장치 3, 스크린 5, 캐비닛 7, 키보드 9, 그리고 마우스 11을 포함하는 컴퓨터 시스템 1을 도시한 것이다. 마우스 11은 그래픽 사용자 인터페이스와 상호작용하기 위한 하나 또는 둘이상의 버튼을 가질 수 있다. 캐비닛 7은 CD-ROM 드라이브 13, 시스템 메모리 그리고 본 발명을 구현하는 컴퓨터 코드를 포함하는 소프트웨어 프로그램, 본 발명과 함께 사용하기 위한 데이터등을 저장하고 이를 꺼내기 위해 사용될 수 있는 하드 드라이브(도 2 참조)를 수용한다. CD-ROM 15는 예시적인 컴퓨터 판독가능 저장매체로써 도시되어 있으나 플로피 디스크, 테이프, 플래시 메모리, 시스템 메모리 및 하드 드라이브를 포함하는 다른 컴퓨터 판독가능 저장매체가 사용될 수도 있다. 또한 캐리어 웨이브에서 실시된 자료신호가(가령 인터넷을 포함하는 네트워크에서) 컴퓨터 판독가능 저장매체일 수도 있다.

도 2 는 본 발명 실시예의 소프트웨어를 실행하기 위해 사용된 컴퓨터 시스템(1)의 시스템 블럭도를 도시한 것이다. 도 1 에서처럼 컴퓨터 시스템(1)은 모니터(3) 및 키보드(9) 및 마우스(11)를 포함한다. 컴퓨터 시스템(1)은 중앙저치리기(51), 시스템 메모리(53), 고정 기억장치(55)(가령 하드 드라이브), 제거가능 기억장치(57)(가령 CD-ROM 드라이브), 디스플레이 어댑터(59), 사운드 카드(61), 스피커(63) 및 네트워크 인터페이스(65)와 같은 서브 시스템을 더욱더 포함한다. 본 발명과 함께 사용하기 위해 적합한 다른 컴퓨터 시스템이 추가의 또는 다소 적은 서브 시스템을 포함할 수도 있을 것이다. 가령 또다른 컴퓨터 시스템은 하나 이상의 처리기(51)(가령 멀티처리기 시스템) 또는 캐시 메모리를 포함할 수도 있다.

컴퓨터 시스템(1)의 시스템 버스 아키텍쳐는 화살표(67)로 표시된다. 그러나 이들 화살표들은 서브 시스템들을 연결시키도록 하는 상호연결기법을 설명하는 것이다. 가령 로컬(지역) 버스는 시스템 메모리 및 표시장치 어댑터로 중앙처리기를 연결시키도록 사용될 수 있다. 도 2 에서 도시된 컴퓨터 시스템은 그러나 본 발명과 함께 사용하기 위해 적합한 컴퓨터 시스템의 한 예이다. 다른 서브 시스템 구성을 갖는 컴퓨터 아키텍처가 또한 사용될 수 있다.

통상적으로 자바 프로그래밍 언어로 기록된 컴퓨터 프로그램은 자바 가상머신에 의해 실행되어지게 되는 바이트 코드 또는 자바 가상머신 지시내로 컴파일된다. 상기 바이트 코드들은 해석을 위해 자바 가상 머신내로 입력되는 클래스 파일속에 저장된다. 도 3 은 인터프리터, 자바 가상머신에 의해 실행을 통해 자바 소스 코드의 간단한 조각의 처리를 도시한다.

자바 소스 코드(101)는 자바로 기록된 Hello World 프로그램을 포함한다. 다음에 상기 소스코드는 소스 코드를 바이트 코드로 컴파일하는 한 바이트 코드 컴파일러(103)내로 입력된다. 상기 바이트 코드들은 이들이 소프트웨어 에뮬레이터된 컴퓨터에 의해 실행될 것이기 때문에 가상 머신 지시들이다. 통상적으로 가상머신 지시들은 범용(즉 어떤 특정 마이크로처리기 또는 컴퓨터 아키텍쳐를 위해 디자인되지 않음)이나 이것이 필요조건은 아니다. 상기 바이트 코드 컴파일러는 자바 프로그램을 위한 바이트 코드를 포함하는 자바 클래스 파일(105)을 출력시킨다.

상기 자바 클래스 파일은 자바 가상머신(107)내로 입력된다. 상기 자바 가상머신은 자바 클래스 파일내 바이트 코드를 해독하고 실행하는 한 인터프리터를 포함한다. 상기 자바 가상머신은 인터프리터인 것으로 간주될 수 있으나 소프트웨어로 마이크로처리기 또는 컴퓨터 아키텍쳐를 에뮬레이트 하기 때문에 가상머신으로 통상 간주된다(가령 하드웨어내에 존재하지 않는 마이크로처리기 또는 컴퓨터 아키텍쳐).

도 4 는 자바 실행시간 시스템의 구현 컴포넌트를 도시한다. 자바 가상머신의 구현은 자바 실행시간 시스템으로써 알려져 있다. 자바 실행시간 시스템(201)은 자바 클래스 파일(203), 스탠다드 내장 자바 클래스(205) 그리고 고유 방법의 입력을 수신하여 자바 프로그램을 실행할 수 있도록 한다. 상기 스탠다드 내장 자바 클래스는 쓰레드 스트링등과 같은 목적을 위한 클래스일 수 있다. 상기 고유 방법들은 자바 프로그래밍 언어가 아닌 프로그래밍 언어로 기록될 수 있다. 상기 고유 방법들은 동적 링크 라이브러리(DLL) 또는 공유 라이브러리내에 저장된다.

상기 자바 실행시간 시스템은 또한 동작시스템(209)과 접속될 수 있다. 가령 입출력 기능은 자바 클래스 파일(203), 스탠다드 내장 자바 클래스(205) 및 고유 방법(207)으로의 인터페이스를 자바 실행시간 시스템으로 제공함을 포함하는 동작시스템에 의해 처리될 수 있다.

한 동적 클래스 적재기 및 검증기(211)는 동작 시스템(209)을 통하여 자바 클래스 파일(203) 및 스탠다드 내장 자바 클래스(205)를 한 메모리(213)내로 적재한다. 추가로 상기 동적 클래스 적재기 및 검증기는 탐지된 어떠한 에러도 보고하면서 자바 클래스 파일내 바이트 코드의 올바름을 검증할 수 있다.

한 고유 방법 링커(215)는 동작 시스템(209)을 통하여 상기 자바 실행시간 시스템내로 고유 방법(207)을 연결시키며 메모리(213)내에 상기 고유 방법을 저장시킨다. 도시된 바와 같이 메모리(213)는 자바 클래스를 위한 한 클래스 및 방법 영역과 고유 방법을 위한 고유 방법 영역을 포함할 수 있다. 메모리(213)내 상기 클래스 및 방법 영역은 불필요 정보가 수집된 히프내에 저장될 수 있다. 새로운 목적이 발생된때 이들은 상기 불필요 정보가 수집된 히프내에 저장된다. 상기 자바 실행시간 시스템은 공간이 더 이상 사용될 수 없는때 불필요 정보가 수집된 히프내 메모리를 다시 요구하기 위해 책임이 있다.

도 4 에 도시된 자바 실행시간 시스템의 중앙에는 실행 엔진(217)이 있다. 상기 실행엔진은 메모리(213)내에 저장된 지시를 수행하며 소프트웨어, 하드웨어 또는 이들의 조합으로 구현될 수 있다. 상기 실행엔진은 객체지향형 응용을 지원하며 동시에 동작하는 다수의 실행엔진이 있으며 각 자바 쓰레드에 하나씩 존재한다. 실행엔진(217)은 지원코드(221)를 사용할 수도 있다. 상기 지원코드는 예외, 쓰레드, 보안등과 관련된 기능들을 제공할 수 있다.

한 자바 프로그램이 실행되는때 기능들이 각 쓰레드내에서 순차적으로 호출된다. 각 쓰레드에 대하여 실행을 끝내지 못한 기능 각각에 대한 프레임을 저장하는 한 실행스택이 있다. 한 프레임이 상기 기능의 실행을 위한 정보를 저장하며 그와 같은 정보가 상태변수, 로컬 변수 및 피연산자 스택을 포함할 수 있다. 한 기능이 호출되는때 그와 같은 기능에 대한 한 프레임이 실행스택상에서 푸쉬된다. 상기 기능이 종료되는때 기능의 프레임은 실행스택으로부터 꺼내진다. 따라서 실행 스택 상위에서의 프레임에 해당하는 기능만이 작동하며 상기 실행 스택의 상위 아래의 프레임에 상응하는 기능들은 상기 기능이 호출될때까지 이들의 실행은 정지된다(즉 종료된다).

도 5 는 한 실행 스택상에 저장된 기능에 대한 프레임을 도시한 것이다. 한 실행 스택(301)이 상기 실행 스택 상부에 있는 한 프레임(303) 그리고 프레임(303) 아래에 저장된 프레임(305 및 307)을 각각 갖는 것으로 도시된다. 스택 포인터 SP 는 상기 실행 스택의 상부를 가리키며 프레임 포인터 FP는 실행 스택(301)의 상부에 있는 프레임내 한 프레임 포인터를 가리킨다.

각 프레임은 상기 프레임에 해당하는 기능을 위한 상태변수, 로컬변수 및 피연산자 스택을 포함하는 것으로 도시된다. 또한 상기 프레임내에 저장된 마지막 항목은 프레임 포인터로써 이는 화살표(309 및 311)로 도시된 상기 실행 스택에서의 현재 프레임 아래의 프레임내 프레임 포인터를 가리킨다.

한 기능이 또다른 기능을 호출하는때 시스템은 먼저 실행 스택(301)에서의 현재 기능을 위한 리턴주소를 푸쉬하며 다음에 최근에 호출된 기능을 위한 새로운 프레임을 푸쉬한다. 이와 같이 하여 새로운 기능이 되돌아오는때 상기 시스템은 상기 실행 스택의 상부에서 프레임을 꺼낼 수 있으며 다음에 리턴주소를 상기 실행 스택으로부터 꺼내고 이같은 리턴주소와 동일한 프로그램 카운터를 고정시키어 호출기능의 실행이 재개될 수 있도록 한다. 또한 이유 프레임(305 및 307)은 리턴주소를 포함하고 활동 프레임(303)은 리턴주소를 포함하지 않는다. 그러나 프레임(303)에 상응하는 기능이 또다른 기능을 호출한다면 상기 리턴주소는 새 기능에 대한 프레임이 상기 실행 스택상에서 푸쉬되기 전에 실행 스택(301)에서 푸쉬되어질 것이다.

프레임 포인터들은 시스템이 정확하게 상기 실행 스택상의 프레임을 가로지를 수 있도록 한다. 가령 스택 포인터 SP 및 프레임 포인터 FP는 실행 스택 상부에서의 프레임을 묘사한다. 또한 프레임(303)에서의 프레임 포인터는 상기 실행 스택에서의 다음 프레임을 명시한다. 프레임(303)에서의 프레임 포인터 바로 아래에 있는 주소는 상기 실행 스택에서의 다음 프레임 시작이며 프레임(303)의 프레임 포인터 내용은 프레임(305)인 실행 스택에서의 다음 프레임내 마지막 항목을 명시한다. 이와 유사하게 프레임(305)에서의 프레임 포인터는 상기 실행 스택에서의 다음 프레임 위치를 명시한다. 따라서 프레임 포인터들의 체인은 시스템이 실행 스택에서의 프레임을 가로지를 수 있도록 한다(가령 프레임들이 상기 실행 스택으로부터 꺼내지는때).

도 5 가 실행 스택의 구현을 도시하고 있으나 본 발명은 도시된 구현으로 제한되지 않는다. 가령 실행 스택은 메모리내에서 상측으로 성장하는 것으로 도시되나 상기 스택이 메모리내에서 하측으로 성장할 수도 있음이 분명하다. 또한 각 프레임내에 저장된 정보는 구현에 따라 변경될 수 있다.

<u>혼합된 실행 스택</u>

도 5 에서 도시된 실행 스택은 같은 프로그래밍 언어로 기록된 기능에 대해서는 잘 수행될 수 있다. 그러나 기능들이 다수의 프로그래밍 언어로 기록된다면 도 5 에서 도시된 실행 스택의 조직은 여러 가지 이유로 만족스럽지 못하다. 가령 다른 프로그래밍 언어내 한 프레임의 포맷은 상당히 다를 수 있다. 따라서 다수의 프로그래밍 언어로 기록된 기능들에 대하여서는 프레임을 통해 프레임 포인터 체인을 발생시키는 것이 가능하지 않을 수 있다. 다른 프로그래밍 언어로 기록된 기능에 대한 프레임의 정확한 크기 또는 내용은 알려져 있지 않을 수도 있다.

이같은 문제를 더욱 명확히 설명할 수 있기 위하여 도 5 에 도시된 실행 스택을 사용하는 한 예를 설명하는 것이 바람직하다. 프레임(305 및 307)이 프레임(303)에 대한 기능이 아닌 다른 프로그래밍 언어로 기록된 기능에 대한 것임을 가정하자. 프레임(303)에 대한 기능이 실행되는때 프레임(305 및 307)의 내용 또는 크기는 알려져 있지 않을 수 있다. 따라서 프레임(303)의 프레임 포인터가 프레임(303)에서와 같은 프로그래밍 언어(도시되지 않음)로 기록된 기능을 위하여 상기 실행 스택상의 이전 프레임으로 정해져 있다면 그와 같은 프레임이 상기 실행 스택에서 어느 곳에서 시작되는가 하는 것이 알려져 있지 않을 수도 있다. 따라서 이들 문제는 다수의 프로그래밍 언어로 기록된 기능을 위해 프레임을 저장하는 한 실행 스택을 구현하는 것을 곤란하게 만든다. 본 발명은 다수의 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 실행 스택을 위한 구현을 제공하며 또한 개선된 자원할당 및 예외처리를 포함하는 다른 장점을 마찬가지로 제공한다.

도 6 은 다수의 프로그래밍 언어로 기록된 기능을 위한 한 실행 스택상의 프레임을 저장하는 높은 수준의 처리를 도시한다. 도 6 에서 도시된 처리는 높은 수준에 있는 것이며 뒤이은 도면 및 설명에서 더욱 상세하게 설명될 것이다. 단계(401)에서 시스템은 한 프로그래밍 언어로 기록된 첫 번째 기능을 위한 실행 스택상의 첫번째 프레임을 저장한다.

상기 첫번째 기능이 또다른 프로그래밍 언어로 기록된 두번째 기능을 호출하는때 시스템은 단계(403)에서 상기 실행 스택상의 첫번째 프레임으로 한 포인터 또는 포인터들을 로컬 기억장치내에 저장시킨다. 바람직한 실시예에서 첫번째 기능의 프로그래밍 언어는 자바 프로그래밍 언어이다. 자바 쓰레드 각각은 쓰레드 로컬 기억장치와 관련돼 있어서 이들 바람직한 실시예에서는 로컬 기억장치가 첫 번째와 두번째 기능을 실행하는 쓰레드를 위한 쓰레드 로컬 기억장치이다. 또한 상기 스택 포인터 및 첫번째 프레임으로의 프레임 포인터는 상기 쓰레드 로컬 기억장치내에 저장된다.

단계(405)에서 시스템은 또다른 프로그래밍 언어로 기록된 두번째 기능을 위한 상기 실행 스택상의 두번째 프레임을 저장한다. 일례로서 또다른 프로그래밍 언어는 C++ 프로그래밍 언어, 파스칼, 포트란, 어셈블리등일 수 있다.

상기 두번째 기능이 첫번째 기능의 프로그래밍 언어로 기록된 세번째 기능을 호출하는때 상기 시스템은 단계(407)에서 실행 스택상의 첫번째 프레임으로의 포인터 또는 포인터들을 포함하는 상기 실행 스택상의 자료블럭(또는 엔트리 프레임)을 저장시킨다. 단계(403)에서 로컬 기억장치내에 저장되었던 포인터 또는 포인터들은 두번째 기능에 대한 두번째 프레임위 실행 스택상에서 푸쉬된 데이터 블럭내로 복사된다. 상기 데이터 블럭은 상기 실행 스택상의 두번째 프레임을 통하여 또는 지나치지 않고 실행 스택을 가로지르기 위한 한 메카니즘을 제공한다. 비록 도 6 에 도시된 처리가 다른 프로그래밍 언어로 기록된 단일 두번째 기능을 도시하고는 있으나 세번째 기능이 호출되기 전에 서로 호출하는 또다른 프로그래밍 언어로 기록된 많은 기능이 있을 수 있음을 이해하여야 한다.

단계(409)에서 시스템은 첫번째 기능의 프로그래밍 언어로 기록된 세번째 기능에 대한 실행 스택상의 세번째 프레임을 저장한다. 따라서 이제 상기 실행 스택은 다수의 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하며 상기 프레임을 통한 실행 스택을 가로지르기 위한 메카니즘을 포함한다. 도 6 에서 도시된 처리를 더욱 설명하기 위하여 자바 프로그래밍 언어 및 또다른 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 한 실행 스택이 도 7 과 관련하여서 설명될 것이다. 도 7 에서 도시된 바와 같이 제어 프레임(451)은 자바 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하며 상기 자바 프로그래밍 언어가 아닌 프로그래밍 언어(가령 C++ 프로그래밍 언어)로 기록된 기능에 대한 외부 프레임을 저장한다. 자바 프레임(453)은 상기 실행 스택의 상부에 저장되며 자바 프레임(455)은 상기 실행 스택상에서 아래에 저장된다. 간단 명료함을 위하여 프레임의 세부사항은 도시되지 않으나 바람직한 실시예에서 상기 프레임들에 대한 세부사항은 도 5 에서 도시된 바와 같다.

한 엔트리 프레임(457)이 상기 실행 스택상의 이전 자바 프레임(459)로의 포인터인 첫번째 LAST_JAVA-SP 및 LAST_JAVA_FP를 저장하는 실행 스택상에 도시된다. 한 외부 프레임(461)은 엔트리 프레임(457)과 자바 프레임(459) 사이의 실행 스택상에 도시된다. 상기 외부 프레임 또는 프레임들은 자바 프로그래밍 언어가 아닌 다른 언어로 기록된 기능을 위한 것이다. 도시된 바와 같이 상기 엔트리 프레임은 한 데이터 블럭으로써 상기 외부 프레임 주변의 적어도 한 포인터를 포함한다.

도 7 에서 도시된 상기 실행 스택을 발생시켰던 프로그램을 실행하는 타임라인에서 자바 프레임(459)에 상응하는 기능이 먼저 실행된다. 자바 프레임(459)에 상응하는 상기 기능은 다른 프로그래밍 언어로 기록된 기능을 호출한다. 이같은 기능에 대한 외부 프레임이 상기 실행 스택상에서 푸쉬되기 전에 시스템은 로컬 기억장치내에 자바 프레임(459)로의 적어도 한 포인터를 저장한다.

뒤이어서 외부 프레임(461)에 상응하는 기능이 자바 기능을 호출한다. 자바 기능을 호출함에 응답하여 상기 시스템은 상기 실행 스택으로 엔트리 프레임(457)을 푸쉬하며 그 속에 적어도 하나의 이 경우에는 LAST_JAVA_SP 및 LAST_JAVA_FP 인 로컬 기억장치내에 저장되었던 자바 프레임(459)으로의 적어도 한 포인터를 저장한다. 다음에 자바 프레임(455)에 상응하는 기능이 자바 프레임(453)에 상응하는 자바 기능을 호출시킨다.

본 발명을 더욱 명확히 이해하기 위하여 도 7 의 실행 스택과 관련하여 설명되어질 네 개의 각기 다른 작용이 있다. 도시된 바와 같이 상기 시스템은 외부 코드를 자바 코드로부터 입력시킬 수 있으며 이는 자바 기능이 다른 프로그래밍 언어로 기록된 외부 기능을 호출하였음을 의미하는 것이다. 또한 상기 시스템은 자바 코드를 외부 코드로부터 입력시킬 수도 있다. 자바 프레임(455)에 상응하는 자바 기능이 되돌아감에 따라 시스템은 자바 코드를 외부 코드로 내보내며 외부 프레임(461)에 상응하는 외부 기능이 되돌아감에 따라 시스템은 외부 코드를 자바 코드로 내보낸다. 다음은 이들 네 개의 처리 각각을 더욱더 상세히 설명한다.

하기 설명은 한 외부기능을 호출하는 한 자바기능으로부터 시작될 것이다. 도 8 은 자바 코드로부터 외부 코드를 입력시키는 처리를 도시한 것이다. 단계(501)에서 시스템은 쓰레드 로컬 기억장치내에 LAST_JAVA_SP 및 LAST_JAVA_FP 로써 현재의 스택 포인터 SP 및 프레임 포인터 FP를 저장한다. 이때 자바 기능은 한 외부 기능을 호출하였음을 기억하여야 한다. 쓰레드 로컬 기억장치내에 저장된 상기 스택 포인터 및 프레임 포인터는 뒤이어서 이들 포인터를 저장하는 데이터 블럭 또는 엔트리 프레임을 사용하여 외부 프레임을 지나치지 않고 횡단하도록 사용될 것이다. 비록 바람직한 실시예는 쓰레드 로컬 기억장치내에 스택 포인터 및 프레임 포인터 모두를 저장하지만 본 발명은 다른 기억장치 위치에 다른 또는 보다 적은 포인터를 저장하도록 사용될 수 있기도 하다.

상기 시스템은 단계(503)에서 외부 코드를 호출한다. 이때 시스템은 상기 실행 스택상의 외부 기능에 대한 한 외부 프레임을 푸쉬할 것이다. 상기 외부 코드를 호출하기 전에 시스템은 상기 실행 스택의 상부에 있는 외부 프레임을 푸쉬하기 전에 실행 스택상의 자바 기능에 대한 리턴주소를 푸쉬한다. 외부 코드에서는 외부 기능이 자바 기능을 호출하거나 자바 기능을 호출하는 다른 외부 기능을 호출할 수 있다.

한 외부 기능을 호출하는 자바 기능이 상기에서 설명되었다. 한 외부 기능으로부터 자바 기능의 호출이 이제 설명될 것이다.

도 9 는 외부 코드로부터 자바 코드를 입력시키는 처리를 도시한 것이다. 통상적으로 상기 외부 기능의 리턴주소가 자바 기능이 호출되기 이전에 상기 실행 스택상에서 푸쉬된다. 단계(551)에서 시스템은 상기 실행 스택상의 엔트리 프레임 또는 데이터 블럭을 푸쉬한다. 쓰레드 로컬 기억장치내에 저장된 LAST_JAVA_SP 및 LAST_JAVA_FP 가 다음에 단계(553)에서 엔트리 프레임내로 복사된다. LAST_JAVA_SP 및 LAST_JAVA_FP 포인터는 상기 실행 스택상에서 이전의 자바 프레임을 가리킨다. 당해분야에 통상의 지식을 가진 자에게 명백한 바와 같이 상기 처리 단계는 본 발명을 설명하고자 제공된 것이며 상기 단계의 어떠한 특정 순서가 요구되어짐을 의미하는 것으로 받아들여져서는 안된다. 가령 본 발명의 실시예는 상기 실행 스택상에서 푸쉬되기 전에 엔트리 프레임내의 한 포인터를 먼저 저장할 수 있으며 그와 같은 실시예는 명백히 본 발명의 범위내에 속하는 것이다.

단계(555)에서 시스템은 쓰레드 로컬 기억장치내에 저장된 LAST_JAVA_SP 및 LAST_JAVA_FP를 제거한다. 쓰레드 로컬 기억장치내 이들 포인터들은 이들 값을 제로로 세트시킴으로써 제거될 수 있으며 바람직한 실시예에서 포인터들은 제로값으로 세트되어 이들 포인터들이 자바 코드 또는 외부코드가 상기 시스템상에서 현재 실행되고 있는가에 대한 결정을 하도록 조사될 수 있다(가령 제로가 아닌=외부 그리고 제로=자바).

종래의 시스템에서는 고유 방법을 위한 자원이 상기 고유 방법이 호출되는때 할당된다. 자주 자바 기능은 다수의 고유 방법을 호출하여 종래의 자원할당이 결국 자원의 다수의 할당 및 할당해제를 일으키도록 할 것이다. 본 발명의 실시예에 따라 자원은 자바 기능이 입력되는때 고유 방법을 위해 할당되어 이들 자원이 자바 기능이 호출할 수 있는 모든 고유 방법에 이용될 수 있도록 한다. 추가로 자원은 자바 기능이 입력되는때 고유 방법을 위해 할당될 수 있으며 자바 기능이 되돌아가는때 할당에서 해제됨으로써 자원할당/할당해제의 효율을 증가시킨다.

단계(557)에서 시스템은 고유 방법을 위한 자원을 할당하며 쓰레드 로컬 기억장치내에 상기 자원을 저장시킨다. 다음에 시스템은 단계(559)에서 자바 코드를 호출한다.

외부 기능에 의해 호출된 자바 기능이 되돌아간 후 자바 기능을 호출하였던 외부 기능과 함께 실행이 계속될 것이다. 도 10 은 외부 코드로 자바 코드를 내보내는 처리를 도시한 것이다. 단계(601)에서 시스템은 쓰레드 로컬 기억장치로 LAST_JAVA_SP 및 LAST_JAVA_FP를 복원시킨다. 상기 시스템은 쓰레드 로컬 기억장치로 상기 엔트리 프레임내에 저장된 포인터를 되복사시킨다. 따라서 외부 기능이 다시 한번 한 자바기능을 호출한다면 포인터는 다시 엔트리 프레임을 만들기 위해 쓰레드 로컬 기억장치내에서 이용될 수 있게 될 것이다.

상기 시스템은 단계(603)에서 고유 방법을 위해 자원을 할당해제시킨다. 바람직한 실시예에서 상기 자원들은 이들이 할당된 후에 쓰레드 로컬 기억장치내에 저장된다. 단계(605)에서 상기 시스템은 상기 실행 스택상에서 리턴주소에 의해 지정된 외부코드로 되돌아간다.

한 자바기능에 의해 호출되었던 외부기능이 되돌아가기만 하면 실행이 상기 외부기능을 호출하였던 자바기능에서 다시 시작될 것이다. 도 11 은 자바코드로 외부코드를 내보내는 처리를 도시한 것이다. 단계(651)에서 상기 시스템은 쓰레드 로컬 기억장치내 LAST_JAVA_SP 및 LAST_JAVA_FP 를 제거시킨다. 바람직한 실시예에서 쓰레드 로컬 기억장치내에 저장된 스택 포인터 및 프레임 포인터는 상기 시스템이 자바 기능을 실행하는때 제로값으로 세트되며 상기 시스템이 외부코드를 실행하는때 제로가 아닌 값으로 세트된다. 상기 스택 및 프레임 포인터는 포인터들이 상기 실행 스택상의 이전 자바 프레임을 가리키는때 외부코드를 실행하는 경우 비제로 값이다.

상기 시스템은 단계(653)에서 자바 코드로 되돌아간다. 상기 시스템은 상기 외부 기능이 호출되는때 상기 실행 스택상에서 푸쉬되었던 리턴주소로 프로그램 카운터를 세트시킴으로써 자바 코드로 되돌아간다.

상기에서는 본 발명의 실시예가 여러 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 한 실행 스택의 구현을 어떻게 제공하는가를 도시 설명하였다. 비록 상기 설명이 자바 프로그래밍 언어에 대한 기능에 대하여 설명되었으나 본 발명은 어떤 특정 프로그래밍 언어로 제한되는 것이 아니다. 본 발명은 다수의 프로그래밍 언어로 기록된 기능을 사용하여 프로그램을 실행하는 시스템에 마찬가지로 적용될 수 있다. 상기 시스템이 상기 실행 스택을 가로지르도록 하고 자원을 보다 효율적으로 할당하도록 함으로써 프로그램이 더욱 효율적으로 실행될 수 있다.

예외

예외란 무언가가 정상상태로부터 벗어나게 됨을 나타내는 신호이다. 가령 한 예외는 시스템이 메모리를 다 사용했거나 파일을 파일 끝이 도달되었음을 나타낼 수 있다. 어떤 예외는 회복할 수 없는 상태(가령 메모리가 다 사용된)를 나타내며 어떤 예외는 회복할 수 있는 상태(가령 파일의 끝에 도달함)를 나타내기도 한다.

한 예외가 발생된때 자바 실행시간 시스템이 그같은 예외를 위한 예외 처리기를 조사한다. 상기 예외가 일어났던 기능내에서 상기 조사가 시작되며 상기 실행 스택상의 기능을 통해 전파된다. 만약 예외 처리기가 발견되면 예외 처리기는 상기 예외를 붙잡고 또다른 예외를 발생시킴을 포함할 수 있는 적절한 액션을 취한다.

일례로써 우리는 자바 가상머신이 C++ 프로그래밍 언어로 기록되는 한 실시예를 설명할 것이다. 따라서 상기 실행 스택은 C++ 프레임 및 자바 프레임을 포함할 것이다. 본 발명의 실시예에서 예외는 상기 실행 스택상의 각기 다른 프레임을 통해 전파될 수 있다. 비록 특정한 실시예가 본 발명을 설명하도록 기재되었으나 본 발명은 어떤 특정 언어 또는 구성으로 제한도지 않는다. 도 12 는 C++ 및 자바 프레임 모두를 포함하는 한 실행 스택을 도시한다. C++ 프레임(703)은 상기 실행 스택의 상부에 위치한다. C++ 프레임(703)에 상응하는 상기 C++ 기능은 자바 프레임(705)에 상응하는 한 자바 기능에 의해 호출되었다. 앞서 설명된 바와 같이 한 엔트리 프레임(707)은 상기 시스템이 C++ 프레임(709 및 711)을 자바 프레임(713)으로 가로지를 수 있도록 하기 위해 상기 실행 스택상에 저장된다.

개념상 C++ 프레임(703)과 자바 프레임(705) 사이에는 한 차폐(715)가 있다. 차폐(715)는 C++ 프레임(703)에서 처리되지 않는 C++ 예외를 붙잡으며 그같은 예외를 자바 프레임(705)으로 보낸다. 상기 예외가 보내지기 전에 C++ 예외를 자바 예외로 변환시키는 것이 필요할 수 있다. 도시된 바와 같이 C++ 프레임(711)과 자바 프레임(713) 사이에는 한 차폐(716)가 있을 수 있다.

자바 프레임(705)(및 엔트리 프레임(707)) 그리고 C++ 프레임(709) 사이에는 한 비차폐(717)가 있다. 상기 비차폐(717)는 자바 프레임(705)내에서 처리되지 않는 자바 예외를 통과시키며 이들을 C++ 프레임(709)으로 보낸다. 도 12 에서 도시된 상기 실행 스택은 본원 명세서에서 간략하게 설명되었으며 자바 코드로 외부 코드를 내보내는 예외의 처리를 설명하는데 유익할 것이다.

도 13A 및 13B 는 자바 코드로 외부 코드를 내보내는 예외의 처리를 도시한다. 단계(801)에서 C++ 예외가 발생되었다. 상기 C++ 예외는 당해분야에서 공지된 바와 같이 적절한 예외 처리기에 의해 붙잡혀지며 처리되어질 수 있다. 만약 C++ 예외가 단계(803)에서 한 예외 처리기에 의해 처리되면 시스템은 상기 예외 처리기에 의해 지시된 바대로 실행을 계속한다.

단계(805) 및 뒤이은 단계(807, 809, 811, 813, 815 및 817)는 도 12의 차폐(715)에 해당된다. 단계(805)에서 시스템은 상기 C++ 프레임내에서 처리되지 않았던 모든 예외를 붙잡는다. 다음에 상기 시스템은 상기 붙잡힌 예외가 단계(807)에서의 자바 예외인 것인가를 결정한다. 다시 말해서 상기 시스템은 C++ 예외가 한 자바 예외로 번역될 수 있는가를 결정한다. 만약 C++ 예외가 한 자바 예외로 변환될 수 없다면 상기 시스템은 단계(809)에서 버스 에러 메시지를 발생시키며 예외를 정지시킨다.

만약 C++ 예외가 한 자바 예외이라면 상기 시스템은 단계(811)에서 상기 예외 스택의 상부로부터 C++ 프레임을 제거시킨다. 상기 C++ 프레임은 상기 예외가 발생된때 실행되는 C++ 기능에 상응하는 것이다. 상기 시스템은 단계(813)에서 쓰레드 로컬 기억장치내에 상기 예외와 한 리턴주소를 저장한다. 상기 저장된 리턴주소는 자바 예외를 발생시키도록 사용될 수 있다.

단계(815)에서 상기 시스템은 한 예외 포워드기로의 한 주소로 상기 실행 스택상의 리턴주소를 정정시킨다. 상기 예외 포워드기는 자바 예외를 발생시키고 상기 실행 스택상의 다음 자바 프레임에 대한 예외 처리기로 점프하는데 관련이 있는 가상 머신기능(C++ 또는 어셈블리로 기록된)의 외부 코드이다.

단계(817)에서 시스템은 외부 코드로 되돌아간다. 상기 자바 프레임(705)에 대한 리턴주소가 상기 예외 포워드기를 참조하기 위해 단계(815)에서 정정되었기 때문에 상기 시스템은 다음으로 도 13B 에서 도시된 예외 포워드기 기능을 수행할 것이다. 도 13B 는 상기 예외 포워드기의 한 실시예를 도시한다. 단계(851)에서 상기 시스템은 한 자바 예외를 만든 것이다. 한 자바 예외는 예외 목적 및 예외가 발생되었던 프로그램 카운터(또는 문제PC) 모두를 포함한다. 프로그램 카운터는 상기 리턴주소로부터 하나를 뺌으로써 쓰레드 로컬 기억장치내에 저장된 리턴주소로부터 용이하게 발생될 수 있다. 비록 설명된 바의 실시예가 리턴주소를 저장하기는 하나 문제 PC는 쓰레드 로컬 기억장치내에 저장될 수 있기도 하다.

쓰레드 로컬 기억장치내에 저장된 리턴주소를 사용하여 예외 포워드기가 리턴주소에 의해 명시된 방법에 대한 상기 예외 처리기 루틴을 얻는다. 적절한 예외 처리기가 식별된 뒤에 단계(855)에서 예외 처리기로 점프가 수행된다. 따라서 C++ 예외가 어떻게 변환되며 자바 예외로써 처리되는가 하는 것이 설명되었다. 다음에는 외부 코드로 자바 코드를 내보내는 예외를 설명할 것이다.

도 14A 및 14B 는 외부 코드로 자바 코드를 내보내는 예외의 처리를 도시한다. 단계(901)에서 자바 예외가 발생되었다. 상기 시스템은 상기 예외가 단계(903)에서 현재의 자바 프레임내에서 처리될 수 있는가를 결정한다. 만약 상기 예외가 이같은 프레임에 대하여 한 예외 처리기에 의해 처리될 수 있다면 상기 프레임은 단계(905)에서 상기 예외 처리기로 점프된다.

만약 자바 예외가 이같은 자바 프레임에 대한 상기 예외 처리기에 의해 처리되지 않는다면 시스템은 단계(907)에서 실행 스택으로부터 자바 프레임을 제거시킨다. 다음에 상기 시스템이 CPU 레지스터내에 저장된 리턴주소에 대한 예외 처리기를 단계(909)에서 발견한다. 단계(911)에서 상기 시스템은 예외 처리기로 점프된다. 상기 예외 처리기는 엔트리 프레임에 대한 상기 실행 처리기가 될 것이다.

도 14B 는 상기 엔트리 프레임의 예외 처리기에 대한 한 처리를 도시한 것이다. 단계(951)에서 상기 예외 처리기는 쓰레드 로컬 기억장치내에 상기 예외를 저장한다. 상기 예외 처리기는 예외를 종료하며 단계(953)에서 C++내 호출 이후에 계속한다. C++는 호출이 있은 뒤 예외가 시작된 때 도 12에서 비차폐(717)로 도시된 비차폐는 만약 한 예외가 단계(955)에서 결정되지 않았는지를 결정한다. 한 예외가 결정되지 않았는지를 알기 위해 로컬 기억장치를 조사함으로써 상기 한 예외가

결정되지 않았는가가 결정될 수 있다. 가령 한 예외가 쓰레드 로컬 기억장치내에 저장되었는가가 결정될 수 있다. 만약 한 예외가 결정되지 않았다면 C++ 예외가 쓰레드 로컬 기억장치내에 저장된 예외를 발생시키므로써 단계(957)에서 C++ 예외가 발생될 수 있다. 따라서 한 자바 예외가 C++ 예외로써 변환되고 다시 발생되었다.

상기에서 설명된 실시예는 정보가 쓰레드 로컬 기억장치내에 저장되어 다수의 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 한 실행 스택을 구현하도록 함을 설명하였다. 도 15 는 쓰레드 로컬 기억장치내에 저장될 수 있는 정보를 도시한 것이다. 상기의 정보는 다른 도면과 관련하여서도 상기에서 설명되었다. 그러나 여기에 저장될 수 있는 정보를 다시 검토하는 것은 유익할 것이다. LAST_JAVA_SP 1001 그리고 LAST_JAVA_FP 1003은 상기 실행 스택상의 이전 자바 프레임으로의 포인터들이다. 이들 포인터들은 상기 실행 스택이 다른 프로그래밍 언어로 기록된 상기 실행 스택상의 프레임을 통하여 또는 이를 지나지 않고 횡단하여 지도록 한다.

자원(1005)은 시스템이 다른 프로그래밍 언어로 기록된 한 기능으로부터 자바 기능으로 들어가는때 할당된다. 다음에 상기 자원들은 자바 기능 또는 어떤 뒤이은 자바 기능에 의해 호출된 어떠한 고유 방법에 의해서도 사용될 수 있다.

한 예외(1007) 및 리턴주소(1009)가 예외가 다른 프로그래밍 언어로 기록된 기능으로부터 (가령 C++ 프로그래밍 언어) 자바 기능으로 예외가 이동할 수 있도록 그리고 그 반대로 이동할 수 있도록 저장된다. 다른 정보가 또한 쓰레드 로컬 기억장치내에 저장될 수도 있다.

발명의 효과

결론

상기에서는 본 발명의 바람직한 실시예에 대한 완전한 설명이 있었으나 수정 또는 동등 실시예가 사용될 수 있기도 한 것이다. 본 발명은 상기에서 설명된 실시예에 대한 적절한 수정을 함으로써 똑같이 적용될 수 있는 것이다. 가령 상기 실시예들은 자바 프레임을 포함하는 혼합된 실행 스택와 관련하여서 설명될 수 있으며 그러나 본 발명의 원리는 다른 시스템 및 언어로도 적용될 수 있는 것이다. 따라서 상기 설명은 본 발명의 범위를 제한하는 것으로 받아들여져서는 아니된다.

도면의 간단한 설명

도 1 은 본 발명 실시예의 소프트웨어를 실행하기 위해 사용될 수 있는 컴퓨터 시스템의 한 예를 도시한 도면.

도 2 는 도 1 의 컴퓨터 시스템의 시스템 블럭도를 도시한 도면.

도 3 은 자바 소스 코드 프로그램이 어떻게 실행되는 가를 도시한 도면.

도 4 는 자바 실행시간 시스템의 구현 컴포넌트를 도시한 도면.

도 5 는 한 실행 스택상에 저장된 기능에 대한 프레임을 도시한 도면

도 6 은 다수의 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장한 한 실행 스택을 구현하는 높은 수준 처리를 도시한 도면.

도 7 은 자바 프로그래밍 언어와 또다른 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 한 실행 스택을 도시한 도면.

도 8 은 자바 코드로부터 외부 코드로 들어가는 처리를 도시한 도면.

도 9 는 외부 코드로부터 자바 코드로 들어가는 처리를 도시한 도면.

도 10 은 외부 코드로 자바 코드를 내보내는 처리를 도시한 도면.

도 11 은 자바 코드로 외부 코드를 내보내는 처리를 도시한 도면.

도 12 는 C++ 프레임과 자바 프레임 사이의 예외 차폐 및 자바 프레임과 또다른 C++ 프레임 사이의 예외 비차폐를 갖는 한 실행 스택을 도시한 도면.

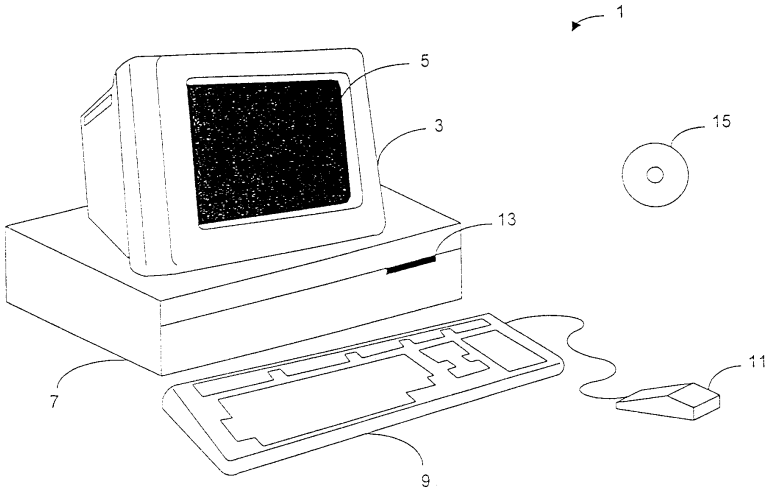
도 13A 및 13B 는 자바 코드로 외부 코드를 내보내는 한 예외의 처리를 도시한 도면.

도 14A 및 14B 는 외부 코드로 자바 코드를 내보내는 한 예외의 처리를 도시한 도면.

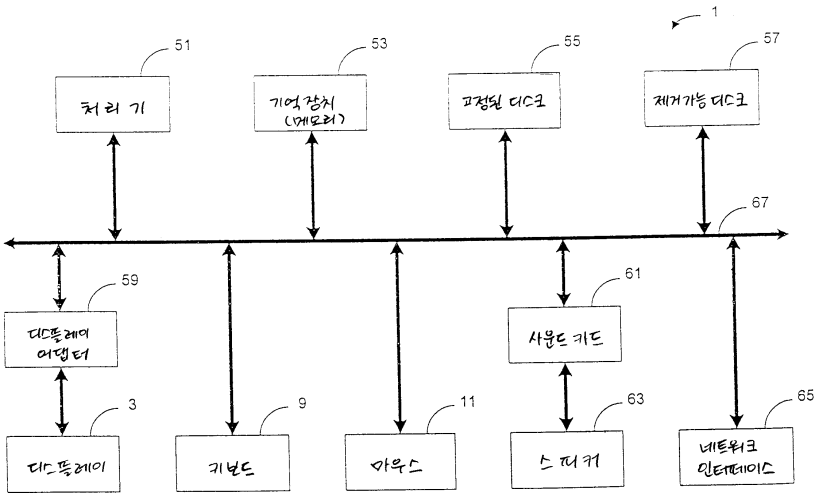
도 15 는 다수의 프로그래밍 언어로 기록된 기능에 대한 프레임을 저장하는 한 실행 스택을 구현하기 위해 쓰레드 로컬 기억장치내에 저장될 수 있는 정보를 도시한 도면.

도면

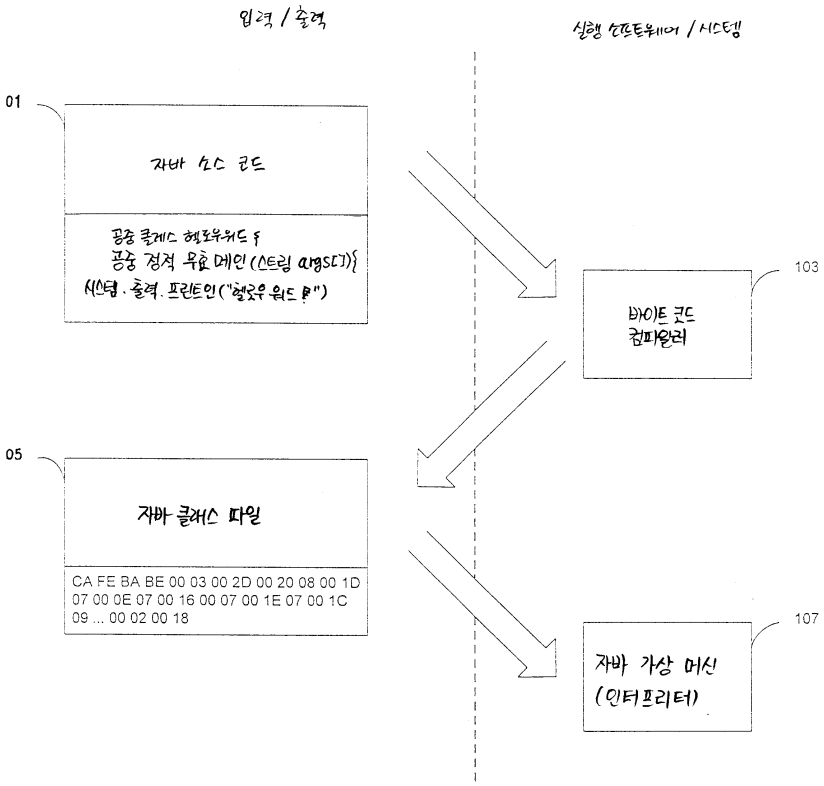
도면1



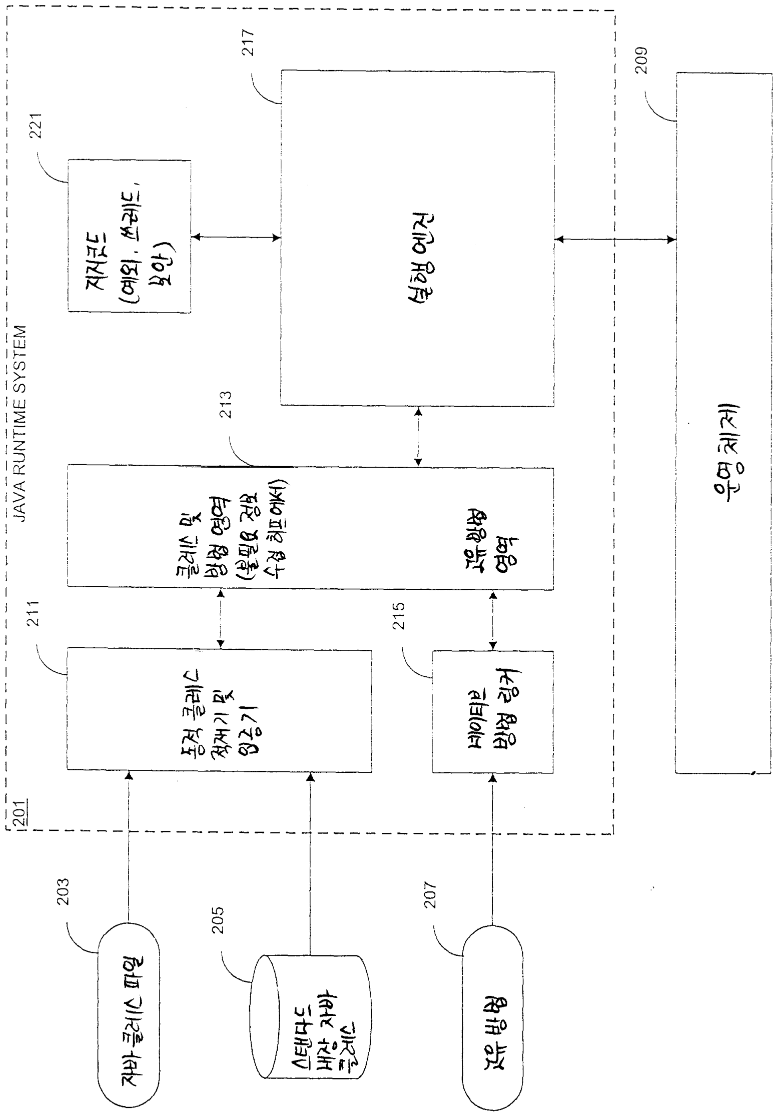
도면2



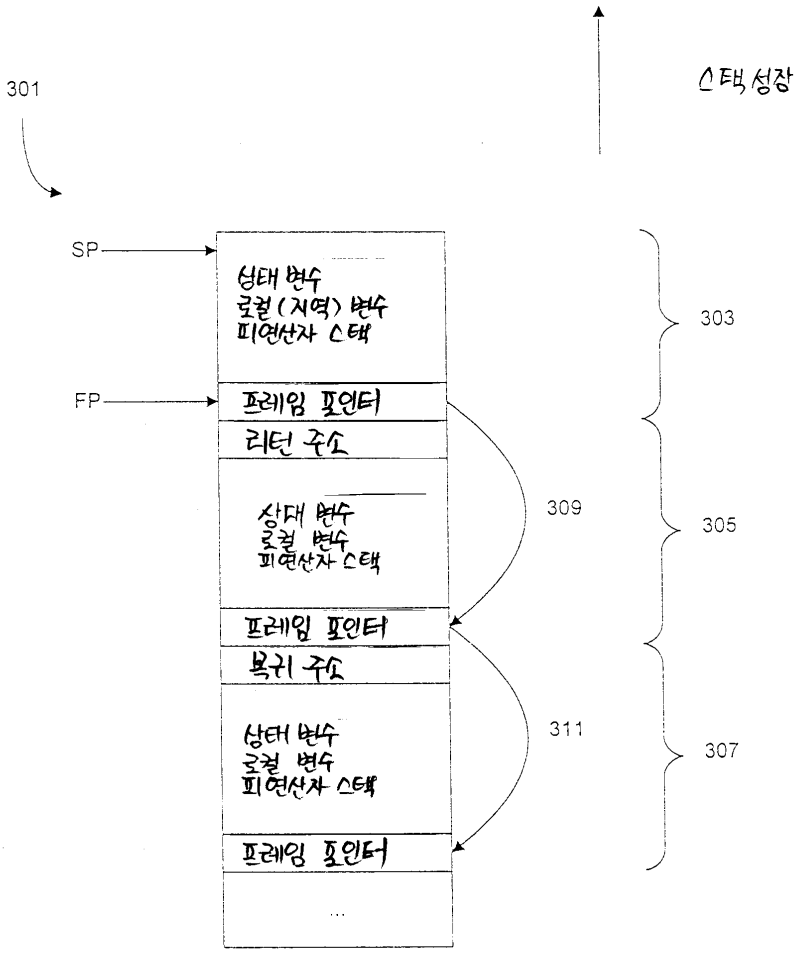
도면3


입력 / 출력
실행 소프트웨어 / 시스템
01
자바 소스 코드
공중 클래스 헬로우워드 {
공중 정적 무효 메인 (스트링 아규스[])
{
시스템.출력.println("헬로우 월드!");
}
}
05
자바 클래스 파일
CA FE BA BE 00 03 00 2D 00 20 08 00 1D
07 00 0E 07 00 16 00 07 00 1E 07 00 1C
09 ... 00 02 00 18
103
바이트 코드
컴파일러
107
자바 가상 머신
(인터프리터)

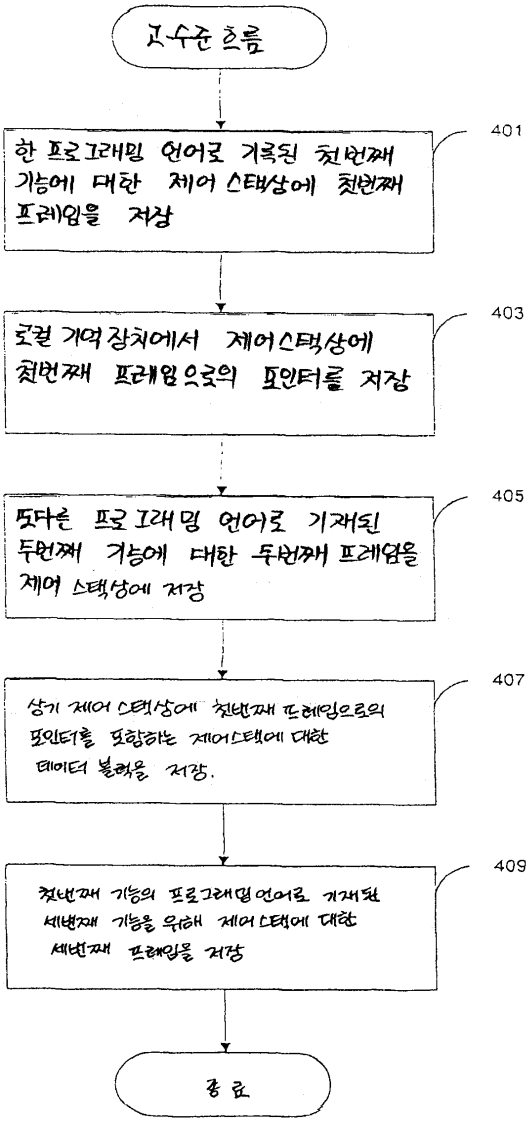
도면4


JAVA RUNTIME SYSTEM
201
203
205
207
209
211
213
215
217
221
운영 체계

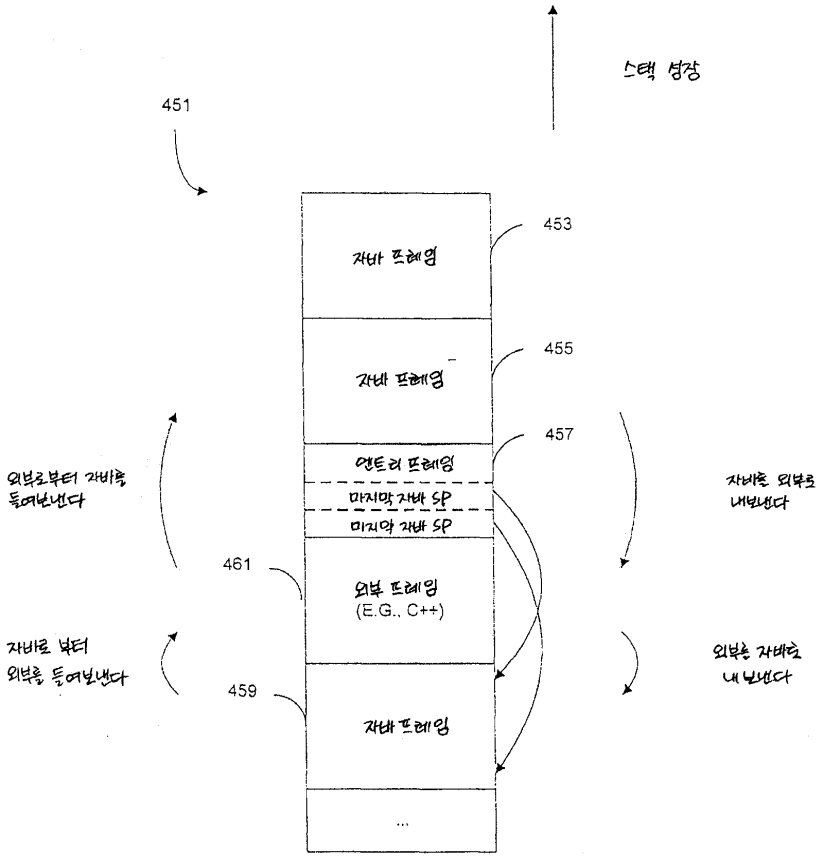
도면5


301
스택 성장
SP
FP
상태 변수
로컬(지역) 변수
피연산자 스택
프레임 포인터
리턴 주소
상태 변수
로컬 변수
피연산자 스택
프레임 포인터
복귀 주소
상태 변수
로컬 변수
피연산자 스택
프레임 포인터
...
303
305
307
309
311

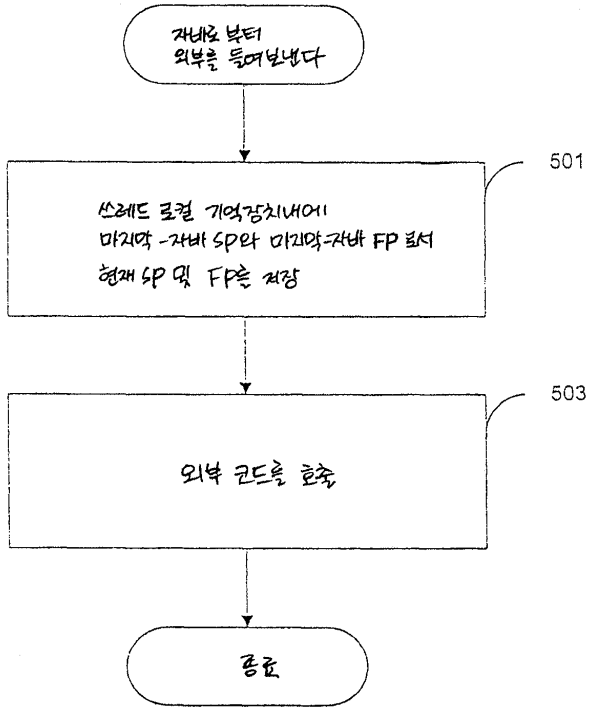
도면6


고수준 흐름
401
한 프로그래밍 언어로 기록된 첫번째 기능에 대한 제어 스택상에 첫번째 프레임을 저장
403
로컬 기억 장치에서 제어스택상에 첫번째 프레임으로의 포인터를 저장
405
또다른 프로그래밍 언어로 기재된 두번째 기능에 대한 두번째 프레임을 제어 스택상에 저장
407
상기 제어 스택상에 첫번째 프레임으로의 포인터를 포함하는 제어스택에 대한 데이터 블록을 저장.
409
첫번째 기능의 프로그래밍언어로 기재된 세번째 기능을 위해 제어스택에 대한 세번째 프레임을 저장
종료

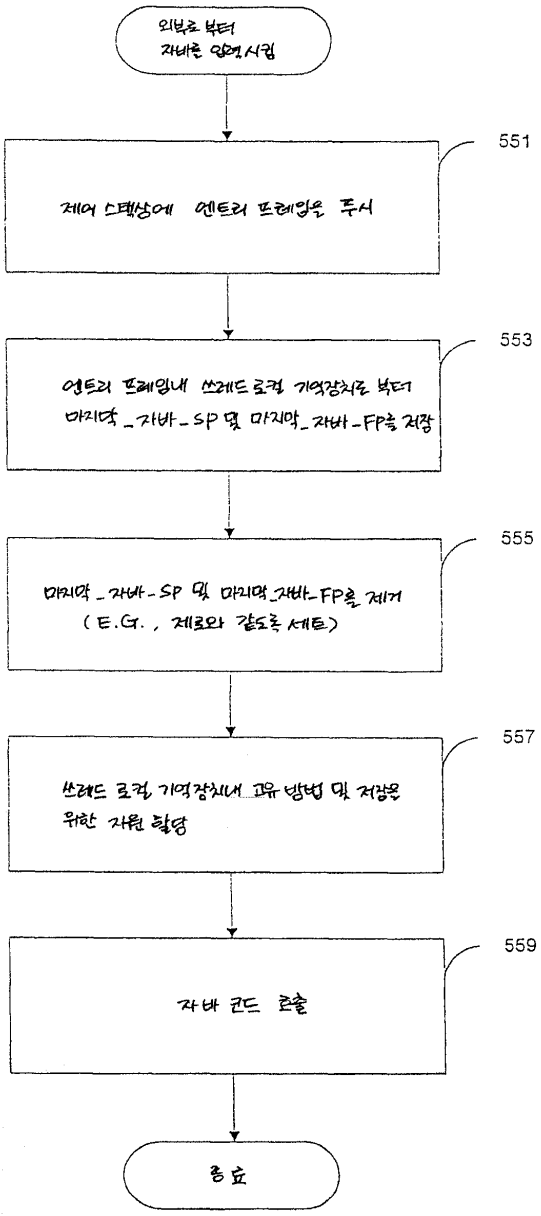
도면7


스택 성장
451
453
455
457
461
459
자바 프레임
자바 프레임
엔트리 프레임
마지막 자바 SP
마지막 자바 SP
외부 프레임
(E.G., C++)
자바 프레임
...
외부로부터 자바를 돌려보낸다
자바로 부터 외부를 돌려보낸다
자바를 외부로 내보낸다
외부를 자바로 내보낸다

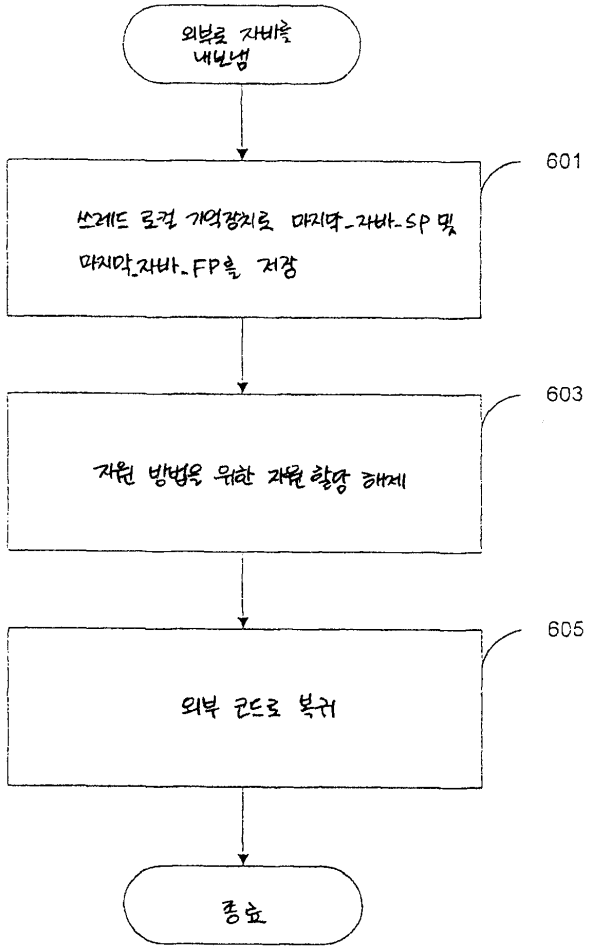
도면8


자바로 부터
외부를 돌려보낸다
501
쓰레드 로컬 기억장치에
마지막 -자바 SP와 마지막-자바 FP 를서
현재 SP 및 FP를 저장
503
외부 코드를 호출
종료

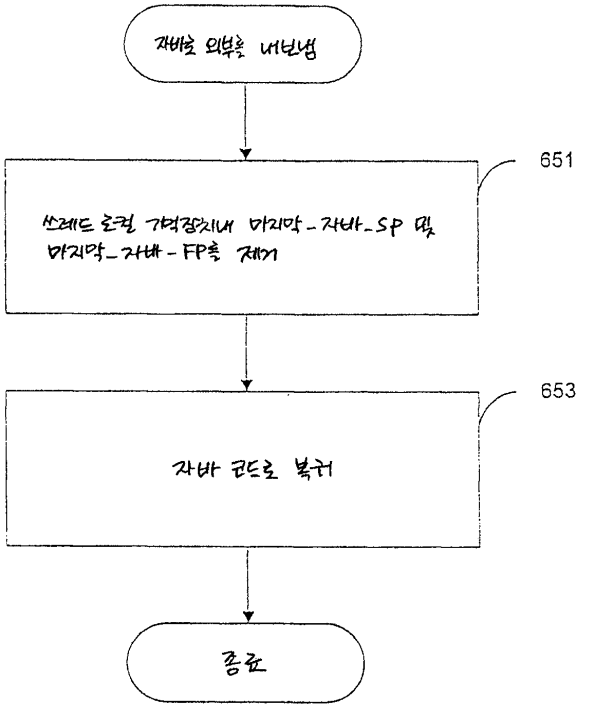
도면9


외부로 부터
자바를 실행시킴
551
제어 스택상에 엔트리 프레임을 푸시
553
엔트리 프레임내 쓰레드 로컬 기억장치로 부터
마지막_자바_SP 및 마지막_자바_FP를 저장
555
마지막_자바_SP 및 마지막_자바_FP를 제거
(E.G., 제로와 같도록 세트)
557
쓰레드 로컬 기억장치내 고유 방법 및 저장을
위한 자원 할당
559
자바 코드 호출
종료

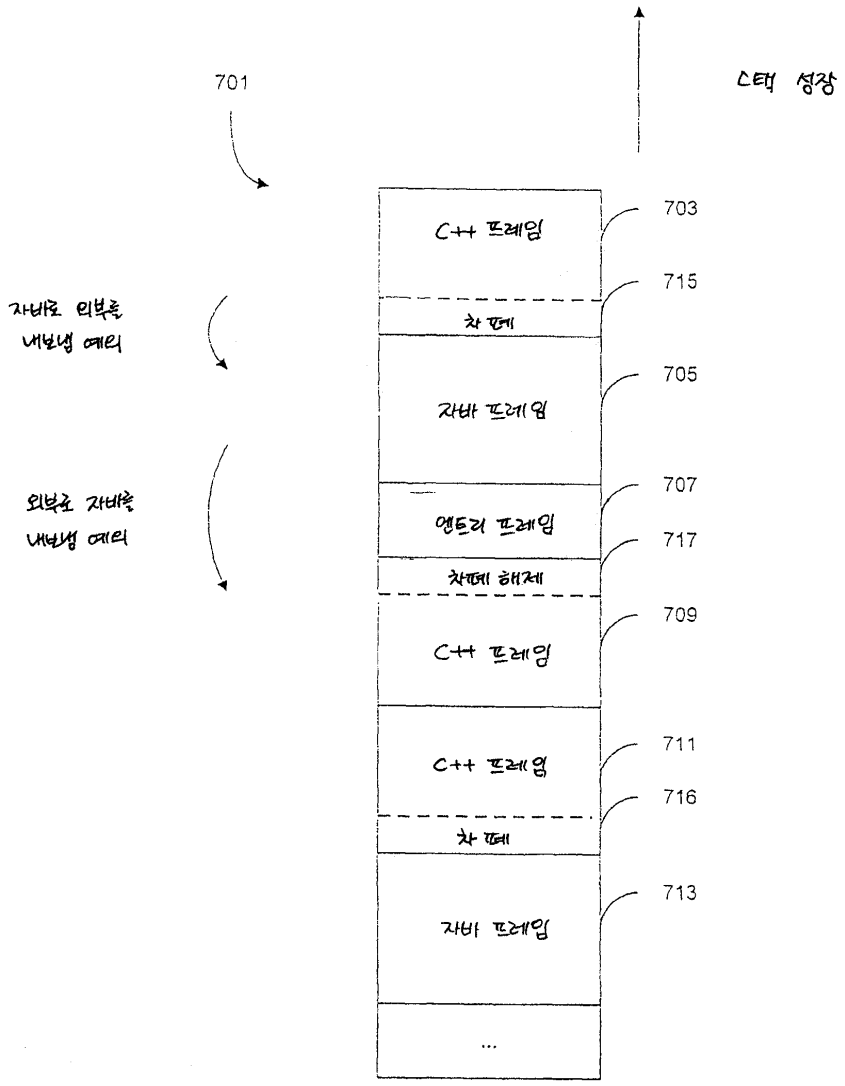
도면10


외부로 자바를 내보냄
601
쓰레드 로컬 기억장치로 마지막_자바_SP 및 마지막_자바_FP를 저장
603
자원 방법을 위한 자원 할당 해제
605
외부 코드로 복귀
종료

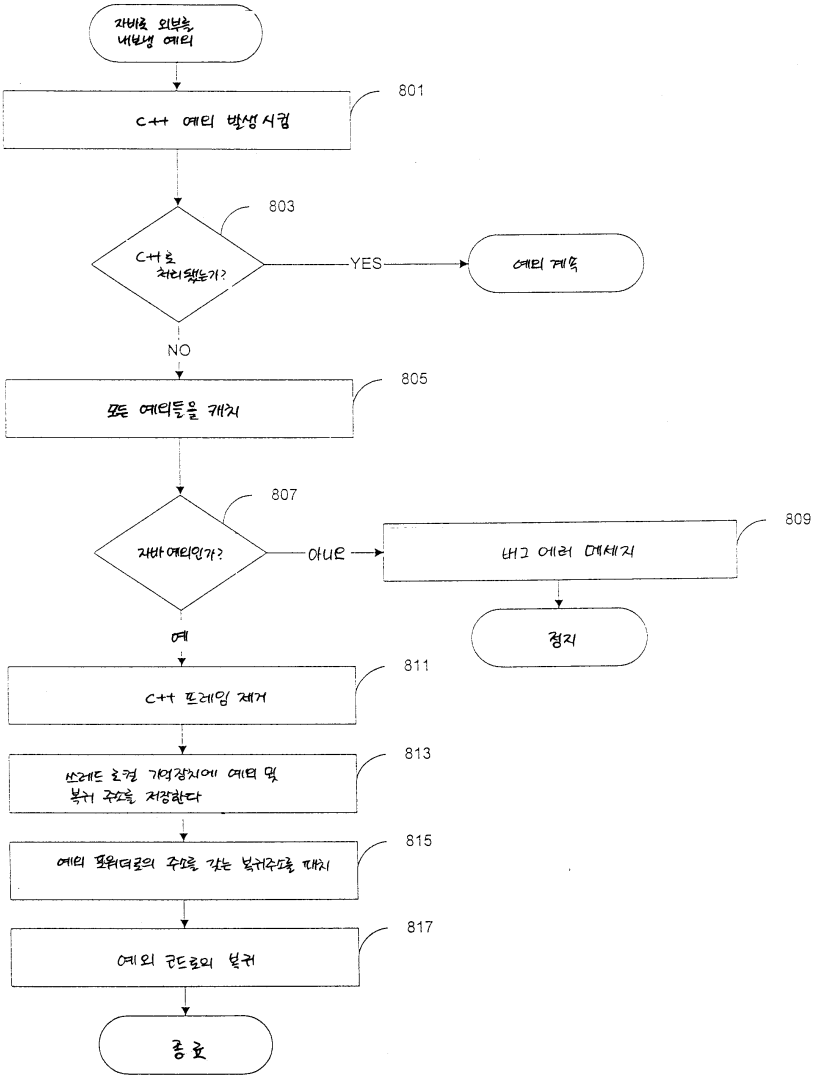
도면11


자바로 외부를 내보냄
651
쓰레드 로컬 기억장치내 마지막_자바_SP 및 마지막_자바_FP를 제거
653
자바 코드로 복귀
종료

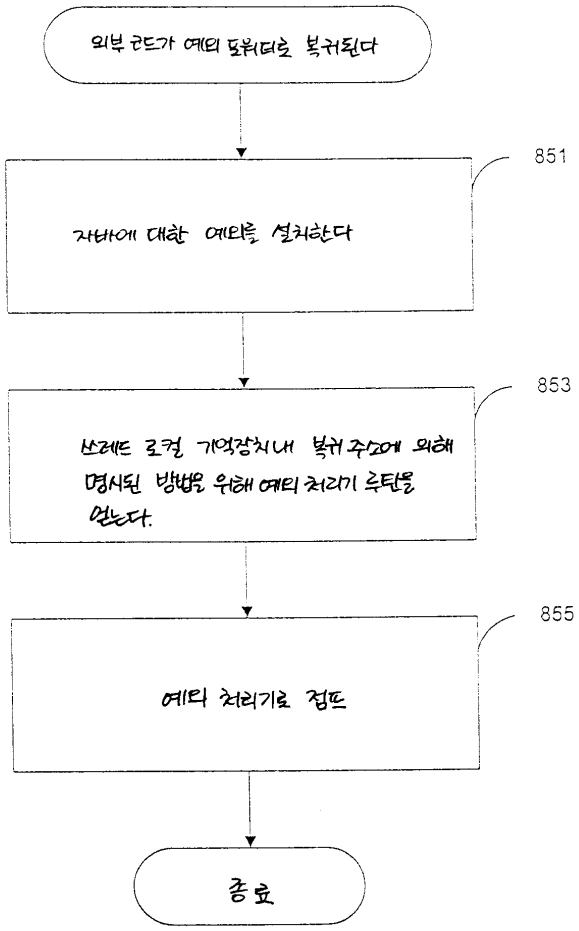
도면12


701
스택 성장
자바로 인보크된 네이티브 메서드
외부로 자바로 네이티브 메서드
C++ 프레임
703
715
자바 프레임
705
엔트리 프레임
707
717
C++ 프레임
709
C++ 프레임
711
716
자바 프레임
713
...

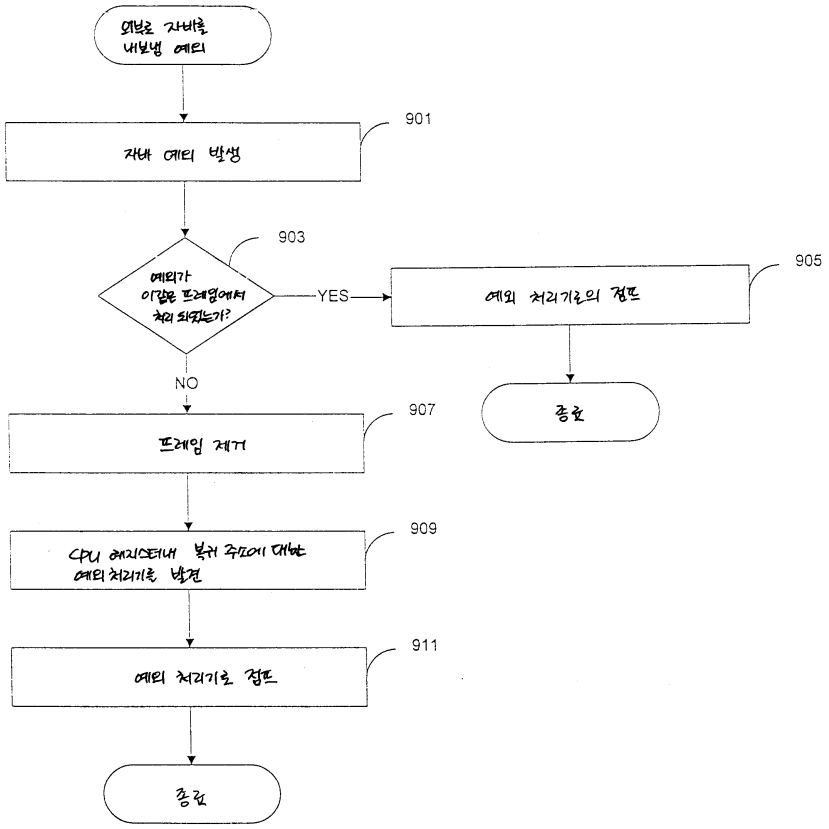
도면13a


자바로 외부로 내보낸 예외
C++ 예외 발생 시점
801
C++로 처리됐는가?
803
YES
예외 계속
NO
모든 예외들을 캐치
805
자바 예외인가?
807
아니오
내그 에러 메세지
809
정지
예
C++ 프레임 제거
811
쓰레드 로컬 기억장치에 예외 및 복귀 주소를 저장한다
813
예외 포워더로의 주소를 갖는 복귀주소를 패치
815
예외 근드로의 복귀
817
종료

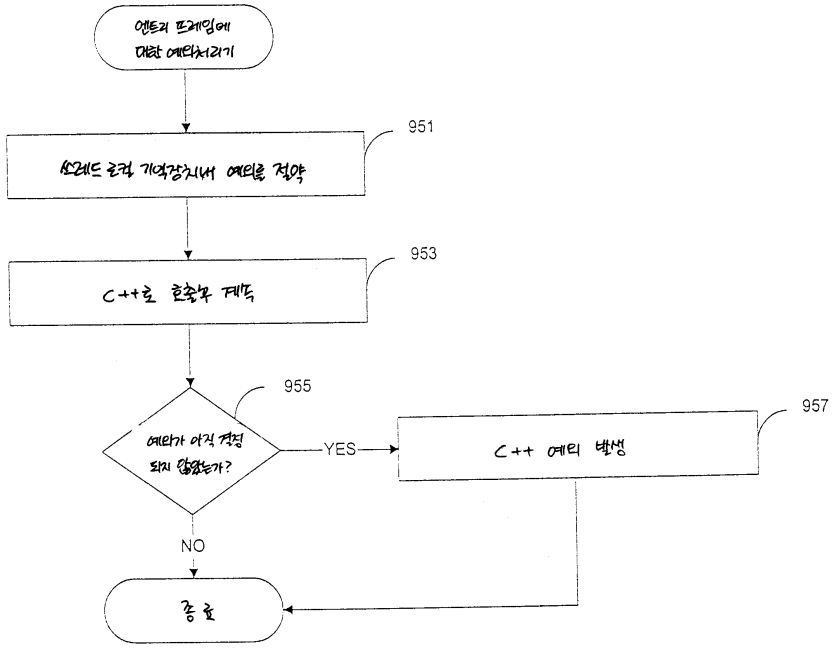
도면13b


외부 코드가 예외 핸들러로 복귀된다
851
자바에 대한 예외를 설치한다
853
쓰레드 로컬 기억장치내 복귀 주소에 의해 명시된 방법을 위해 예외 처리기 목록을 얻는다.
855
예외 처리기로 점프
종료

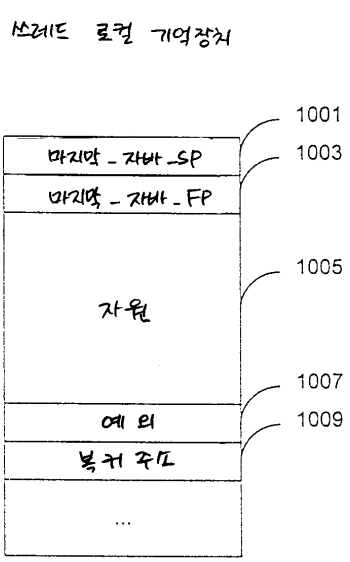
도면14a


외부로 자바를 내보냄 예외
자바 예외 발생
901
예외가 이절로 프레임에서 처리 되었는가?
903
YES
예외 처리기로의 점프
905
NO
종료
프레임 제거
907
CPU 레지스터내 복귀 주소에 대한 예외처리기를 발견
909
예외 처리기로 점프
911
종료

도면14b


엔트리 프레임에 대한 예외처리기
쓰레드 로컬 기억장치내 예외를 점유
951
C++로 흔출부 제독
953
예외가 아직 정의 되지 않았는가?
955
YES
C++ 예외 발생
957
NO
종료

도면15


쓰레드 로컬 기억장치
1001
마지막 - 자바 - SP
1003
마지막 - 자바 - FP
1005
자원
1007
예 외
1009
복귀 주소
...