

(19) 대한민국특허청(KR) (12) 등록특허공보(B1)

(51) 。 Int. Cl. 6
G06F 11/28

(45) 공고일자 2002년05월27일
(11) 등록번호 10 - 0338223
(24) 등록일자 2002년05월14일

(21) 출원번호 10 - 1999 - 0006338
(22) 출원일자 1999년02월25일

(65) 공개번호 특1999 - 0077479
(43) 공개일자 1999년10월25일

(30) 우선권주장 09/052,329 1998년03월31일 미국(US)

(73) 특허권자 인터내셔널 비지네스 머신즈 코포레이션
포만 제프리 엘
미국 10504 뉴욕주 아몬크

(72) 발명자 알렉산더3세월리엄프레스톤
미합중국78759텍사스주오스틴갤럭시아드라이브3904
베리로버트프란시스
미합중국78759텍사스주오스틴카르던드라이브8107
멀베이도날드엘.
미합중국78664텍사스주라운드록윌더니스웨이31
유르큐하르트로버트존
미합중국78759텍사스주오스틴미스틱옥스트레일9210

(74) 대리인 김성기
송병옥

심사관 : 임영희

(54) 데이터 처리 시스템 및 애플리케이션의 구조화된 프로파일링 방법 및 장치

요약

본 발명은 프로그램의 메모리 성능을 모니터링하는 방법을 개시한다. 주기적으로 발생하는 이벤트(event)가 검출되면, 주기적으로 발생하는 이벤트의 검출에 응답하여 프로그램과 연관된 스택(stack)이 식별되는데, 여기서 이벤트는 타이머 인터럽트(timer interrupt)가 될 수 있다. 스택을 검사하여 프로그램과 연관된 현재 실행중인 각 루틴 (및 특정 호출 포인트, 또는 오프셋)을 식별한다. 각 루틴 (및 오프셋)은 트리 구조의 노드(node)로 표시된다.

대표도

도 1

색인어

데이터 처리 시스템, 구조화된 프로파일링, 호출 스택, 트리 구조, 메모리 액세스, 페이지 폴트, 인스트럭션 폴트, 샘플 x트리 보고서

명세서

도면의 간단한 설명

- 도 1은 본 발명이 구현될 수 있는 데이터 처리 시스템의 블록 다이어그램.
- 도 2는 본 발명의 바람직한 실시예에 따른 호출 스택(call stack)의 다이어그램.
- 도 3은 본 발명의 바람직한 실시예에 따른 호출 스택 샘플을 예시하는 도면.
- 도 4a는 전체 호출의 추적(trace)을 예시하는 다이어그램.
- 도 4b는 본 발명의 바람직한 실시예에 따라 도 4a에 도시된 실행 플로우의 특정 타이머 기반 샘플링을 도시한 도면.
- 도 5는 본 발명의 바람직한 실시예에 따른 트리의 다이어그램.
- 도 6은 본 발명의 바람직한 실시예에 따라 호출 스택 트리를 생성하는 프로세스의 플로우차트.
- 도 7은 본 발명의 바람직한 실시예에 따라 샘플링이 이루어지는 동안 얻어진 어드레스로부터 기능을 식별하기 위한 프로세스의 플로우차트.
- 도 8은 본 발명의 프로세스를 사용하여 얻어진 구조화된 프로파일의 다이어그램.
- 도 9는 본 발명의 프로세스를 사용하여 생성된 기록의 다이어그램.
- 도 10은 본 발명의 프로세스를 사용하여 생성될 수 있는 또 다른 형태의 보고서(report)의 다이어그램.
- 도 11은 본 발명의 바람직한 실시예에 따라 메모리 액세스를 분석하는 프로세스의 플로우차트.
- 도 12는 본 발명의 바람직한 실시예에 따른 페이지 폴트(page fault)에 응답하여 얻어진 어드레스로부터 기능을 식별하는 프로세스의 플로우차트.
- 도 13은 본 발명의 프로세스를 사용하여 얻어진 메모리 프로파일의 다이어그램.
- 도 14는 본 발명의 바람직한 실시예에 따른 인스트럭션 폴트(instruction fault)에 대한 샘플 x트리 보고서.
- 도 15는 본 발명의 바람직한 실시예에 따른 데이터 폴트에 대한 샘플 x트리 보고서.

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 개선된 데이터 처리 시스템에 관한 것으로, 구체적으로는 데이터 처리 시스템의 성능을 모니터링(monitoring)하기 위한 개선된 방법 및 장치에 관한 것이다. 좀더 구체적으로, 본 발명은 데이터 처리 시스템 및 데이터 처리 시스템 내에서 실행 중인 애플리케이션의 구조화된 프로파일링을 위한 개선된 방법 및 장치를 제공한다.

데이터 처리 시스템의 성능을 분석하여 향상시키고, 그 데이터 처리 시스템 내에서 애플리케이션을 실행하는데는 데이터 처리 시스템 내에서 어느 소프트웨어 모듈이 시스템 자원을 사용하고 있는지를 아는 것이 도움이 된다. 데이터 처리 시스템을 효율적으로 관리하고 개선하기 위해서는 여러 가지 시스템 자원이 언제 어떻게 사용되는지를 알 필요가 있다. 여러 가지 소프트웨어 애플리케이션이 데이터 처리 시스템 내에서 실행 중일 때 자원의 소비 정도를 결정하기 위해서는 데이터 처리 시스템을 모니터링하여 검사할 수 있는 성능 툴이 사용된다. 예를 들어, 성능 툴은 데이터 처리 시스템 내에서 가장 빈번하게 실행되는 모듈 및 인스트럭션을 식별하거나, 가장 큰 양을 차지하는 메모리를 할당하거나 가장 많은 I/O 요구를 수행하는 모듈을 식별할 수 있다. 하드웨어 성능 툴은 처음부터 시스템 내에 내장되거나 또는 시간적으로 나중에 추가될 수 있다. 소프트웨어 성능 툴은 또한 개인용 컴퓨터 시스템과 같은 데이터 처리 시스템에 사용이 가능한데, 이 경우 데이터 처리 시스템은 통상적으로 내장형(built-in) 하드웨어 성능 툴을 포함하고 있지 않으며, 혹시 포함하고 있다 하더라도 그 수가 많지 않다.

소프트웨어 성능 툴 중의 하나로 추적 툴(trace tool)이 알려져 있으며, 이 추적 툴은 소정의 이벤트가 발생할 때 그 이벤트를 로그하여 특정 인스트럭션 시퀀스를 계속 추적한다. 예를 들어, 추적 툴은 모든 진입부(entry)를 모듈, 서브루틴, 방법, 기능 또는 시스템 구성 요소 내로 로그(log)하고, 모든 출구부(exit)를 모듈, 서브루틴, 방법, 기능, 또는 시스템 구성 요소로부터 로그할 수 있다. 이와 달리, 추적 툴은 요구자 및 각 메모리 할당 요구에 대해 할당된 메모리의 양을 로그할 수 있다. 통상적으로, 각 이벤트마다 시간 스탬프가 찍힌 기록이 만들어진다. 진입부-출구부 기록과 유사한 기록 쌍은 또한 잠금 요구 및 해제, I/O 또는 데이터 전송의 개시 및 종료를 기록하기 위해 임의의 코드 세그먼트의 실행을 추적하는데 사용될 뿐만 아니라, 다른 여러 가지 해당 이벤트에 대해서도 사용된다.

또 다른 사용 툴은 프로그램 핫 스팟(program hot spot)과 같은 이벤트를 식별하기 위한 프로그램 샘플링을 포함하고 있다. 이러한 기법은 애플리케이션 또는 데이터 처리 시스템 실행을 일정한 간격으로 인터럽트한다는 아이디어에 기반을 두고 있다. 매 인터럽트마다, 프로그램이거나 또는 더 큰 프로세스의 일부를 이루는 프로세스인, 현재 실행 중인 스레드(thread)의 프로그램 카운터가 기록된다. 통상적으로, 사후 처리 시점에서, 상기 툴은 데이터 처리 시스템에 대한 적재 맵(load map)과 심볼 테이블 정보에 대해 분해되는 값을 캡처(capture)하고, 현재 시간이 소비되고 있는 위치에 대한 프로파일링이 상기 분석으로부터 얻어진다.

이벤트에 기초한 프로파일링은 여러 가지 단점을 가지고 있다. 예를 들어, 이벤트에 기초한 프로파일링은 성능면(매 진입부당 이벤트, 매 출구부당 이벤트)에서 가격이 비싸며, 이로 인해 최종 성능 뷰(view)를 예측하는 것을 방해할 수 있으며, 실제로도 흔히 방해한다. 또한, 이러한 기법은 코드 내로 진입부/출구부 이벤트를 정적 또는 동적으로 삽입해 주어야 하므로 이러한 기법이 항상 사용 가능한 것은 아니다. 이러한 이벤트 삽입은 때로 불가능하거나, 적어도 상당히 어렵다. 예를 들어, 소스 코드가 향후 구현될 코드용으로 사용하는 것이 불가능한 경우, 이벤트에 기초한 프로파일링은 실행이 불가능할 수 있다.

반면에, 샘플에 기초한 프로파일링은 시스템 성능 뷰('평면 뷰')를 제공하지만, 비용이 감소되고 후크(hook) 성능에 대한 의존도가 감소되는 장점을 제공한다.

또한, 샘플에 기초한 프로파일링 기법은 외관상으로 관련이 없어 보이는 다수의 작은 기능에 있어서 시간이 소비되는 위치 또는 핫 스팟이 명확하게 나타나지 않는 상황에서 시간이 소비되는 위치를 식별하지 못한다. 프로그램 구조를 이해하지 못하면, 상기 '평면' 프로파일로는 성능 개선이 이루어질 수 있는 위치에 대한 결정 방법이 명확하지 않다.

발명이 이루고자 하는 기술적 과제

따라서, 데이터 처리 시스템 및 그 데이터 처리 시스템 내에서 실행되는 애플리케이션을 프로파일링하는 개선된 방법 및 장치를 구비하는 것이 효과적이다.

본 발명의 목적은 개선된 데이터 처리 시스템을 제공하는 것이다.

본 발명의 또 다른 목적은 개선된 데이터 처리 시스템의 성능을 모니터링하는 개선된 방법 및 장치를 제공하는 것이다.

본 발명의 또 다른 목적은 데이터 처리 시스템 및 그 데이터 처리 시스템 내에서 실행되는 애플리케이션의 구조화된 프로파일링에 대한 개선된 방법 및 장치를 제공하는 것이다.

발명의 구성 및 작용

본 발명은 하나 이상의 활성 프로그램을 갖는 전체 시스템의 성능을 모니터링하는 방법을 제공한다. 주기적으로 발생하는 이벤트가 식별되고, 이벤트(예컨대, 타이머 인터럽트) 발생시의 활성 스레드와 연관되는 호출 스택이 얻어진다. 호출 스택 내의 각 루틴은 트리 구조를 갖는 계정 데이터 구조(accounting data structure) 내의 노드로 표시된다.

또한, 각 스레드의 실행과 관련된 정보는 루틴을 표시하는데 사용되는 노드, 또는 루틴 내의 소정의 오프셋과 연관을 가질 수 있다. 현재 실행 중인 각 스레드를 식별하는 보고서가 작성된다. 본 발명은 또한 데이터 처리 시스템에 있어서 그 데이터 처리 시스템 내의 메모리 사용을 분석하는 방법을 제공한다. 메모리 액세스가 발생할 때마다 페이지 폴트가 초기화된다. 모니터링이 수행되어 데이터 처리 시스템 내의 페이지 폴트를 검출한다. 페이지 폴트와 연관되는 호출 스택이 페이지 폴트의 검출에 응답하여 식별된다. 호출 스택이 검사되어 페이지 폴트가 검출되었을 때에 실행 중인 각 루틴을 식별한다. 또한, 각 루틴은 트리 구조의 노드로 표시된다.

도면을 참조하여 설명하면, 본 발명이 구현할 수 있는 데이터 처리 시스템(100)의 블록도가 도 1에 도시되어 있다. 데이터 처리 시스템(100)은 주변 장치 연결부(peripheral component interconnect; PCI)를 사용한다. 예시된 예에서는 PCI 버스를 사용하고 있지만, 마이크로채널(Micro Channel)과 ISA와 같은 다른 버스 구조를 사용해도 된다. 프로세서(102) 및 주 메모리(104)는 PCI 브리지(108)를 통해 PCI 로컬 버스(106)와 연결되어 있다. PCI 브리지(108)는 또한 집적화된 메모리 제어 장치 및 프로세서용 캐시 메모리를 포함할 수 있다. PCI 로컬 버스(106)에 대한 추가 연결이 구성 요소의 직접적인 상호 연결 또는 애드인 기판(add-in board)을 통해 이루어질 수 있다. 도시된 예에서는, 근거리 통신망(LAN) 어댑터(110), SCSI 호스트 버스 어댑터(112) 및 확장 버스 인터페이스(114)가 구성 요소의 직접 연결에 의해 PCI 로컬 버스(106)에 연결된다. 반면에, 오디오 어댑터(116), 그래픽 어댑터(118) 및 오디오/비디오 어댑터(119)(A/V)는 확장 슬롯으로 삽입되는 애드인 기판에 의해 PCI 로컬 버스(106)에 연결된다. 확장 버스 인터페이스(114)는 키보드 및 마우스 어댑터(120), 모뎀(122) 및 추가 메모리(124)에 대한 연결을 제공한다. 예시된 실시예에서, SCSI 호스트 버스 어댑터(112)는 하드 디스크 드라이브(126), 테이프 드라이브(128) 및 CD-ROM(130)에 대한 연결을 제공한다. 통상적인 PCI 로컬 버스의 실시예는 3개 또는 4개의 PCI 확장 슬롯 또는 애드인 커넥터를 지원한다. 도시된 실시예는 마더보드 상의 4개의 로드와 3개의 확장 슬롯을 포함한다. 당업자는 도 1의 하드웨어에 대한 변경이 가능하다는 것을 이해할 것이다. 예를 들어, 광디스크 장치 등과 같은 다른 기타 주변 장치가 도 1에 도시된 하드웨어에 추가되거나 또는 하드웨어 대신 사용될 수 있다. 도시된 실시예가 본 발명에 대한 아키텍처를 제한하도록 의도하는 것은 아니다. 예를 들어, 본 발명은 메인프레임 컴퓨터 상에서 구현될 수 있다.

본 발명은 데이터 처리 시스템 및 데이터 처리 시스템 내에서 실행 중인 애플리케이션의 구조화된 프로파일링을 위한 시스템, 방법 및 컴퓨터 판독 가능 매체를 제공한다. 본 발명의 일 실시예는 다수의 운영 체제에서 현재 사용 가능한 타이머 인터럽트에 의해 스레드가 인터럽트되는 인터럽트된 스레드의 스택으로부터 정보를 입수한다. 상기 타이머 인터럽트는 호출 스택으로부터 정보를 샘플링하는데 사용된다. 호출 스택을 역탐지(walk back up)함으로써 분석에 필요한 전체 호출 스택이 얻어진다. 이러한 호출 스택은 인터럽트 발생 시점에서 기능/방법 호출의 시퀀스를 기록한다. 호출 스

택(즉, 모듈, 기능, 방법 등)은 루틴의 순차 목록 + 소정 프로그램의 실행 중에 입력된 루틴 내의 오프셋이다. 예를 들어, 프로세서가 루틴 C에서 인스트럭션을 실행하고 도중에 루틴 A가 루틴 B를 호출하며, 이어서 루틴 B가 루틴 C를 호출하면, 호출 스택은 ABC가 된다. 제어가 루틴 C로부터 루틴 B로 복귀하면, 호출 스택은 AB이다.

따라서, 처리 도중 또는 처리 후에, 생성된 프로파일은 현재까지 알려진 프로그램 카운터 샘플링 기법의 경우와 같이 단지 가능한 호출 스택의 잎(leaf)들만이 아니라 가능한 모든 호출 스택의 샘플링을 나타낸다. 잎들은 호출 스택 트리 구조에서 1차 노드라고도 불리우는 루트 노드(root node)로부터 가장 멀리 떨어진 노드이다. 바꿔 말하면, 잎은 브랜치의 끝부분에 있는 노드로 후손(descendent)이 없는 노드이다. 후손은 모 노드(parent node)의 자(children)이고, 잎은 자가 없는 노드이다.

도 2를 참조하면, 본 발명에 따른 바람직한 실시예의 호출 스택의 다이어그램이 도시되어 있다. 스택은 소정의 프로그램 또는 프로그램들이 절차(procedure) 및 기능 호출 어드레스와 같은 상태 데이터, 전달된 파라미터 및 때로는 로컬 변수를 저장하는 예약 메모리의 소정 영역이다. 스택 프레임은 단일 기능 호출(invocation)용 로컬 저장장치[인수(argument), 복귀 어드레스 복귀 값, 로컬 변수]를 표시하는 스레드 스택의 일부이다. 모든 활성 상태의 실행 스레드는 자신의 스택 공간용으로 할당된 시스템 메모리의 일부를 갖는다. 스레드의 스택은 스택 프레임의 시퀀스로 이루어져 있다. 스레드 스택 상의 프레임 세트는 항상 해당 스레드의 실행 상태를 나타낸다. 스택 프레임은 통상적으로 상호 연결(internlink)되어 있기 때문에(예컨대, 각 스택 프레임은 이전 스택 프레임의 위치를 결정함), 흔히 스택 프레임의 시퀀스를 추적하여 호출 스택을 형성하는 것이 가능하다. 호출 스택은 모든 미완료 기능 호출(not-yet-completed function calls)을 표시하는데, 다시 말하면, 임의의 시점에서 기능 호출 시퀀스(function invocation sequence)를 나타낸다.

호출 스택(200)은 정보를 포함하는데, 이 정보는 메인 프로그램까지 진행되는 동안 현재 실행 중인 루틴 및 정보를 호출한 루틴 등을 식별한다. 호출 스택(200)은 다수의 프레임(202, 204, 206, 208)을 포함한다. 예시된 실시예에서, 스택 프레임(202)은 호출 스택(200)의 최상부에 있고, 스택 프레임(208)은 호출 스택(200)의 최하부에 위치한다. 호출 스택의 최상부는 또한 루트(root)라고도 불리운다. (대부분의 운영 체제에서 사용되는) 타이머 인터럽트는 인터럽트된 스레드의 프로그램 카운터값(pcv: program counter value) 및 현재 활성 상태에 있는 해당 스레드에 대한 스택 프레임을 얻도록 수정된다. 인텔(Intel) 아키텍처에서, 이것은 통상적으로 레지스터의 내용, 즉 EIP(프로그램 카운터) 및 EBP(스택 프로그램에 대한 포인터)에 의해 표시된다. 현재 활성 상태인 스택 프레임을 액세스함으로써, (통상적인) 스택 프레임 연결 규약을 이용하여 모든 프레임을 체인화할 수 있다. 표준 연결 규약의 일부는 또한 기능 복귀 어드레스가 호출된 기능의 스택 프레임 바로 상부에 위치되도록 지시하는데, 이것은 호출된 기능에 대한 어드레스를 확인하는데 사용된다. 상기에서는 인텔 기반 아키텍처를 채용하는 것으로 설명하였지만, 본 실시예는 이에 제한되는 것은 아니다. 대부분의 아키텍처가 수정된 프로파일링 인터럽트 핸들러에 의해 유사하게 탐색될 수 있는 연결 규약을 채용하고 있다.

타이머 인터럽트가 발생할 경우, 얻어진 제1 파라미터는 프로그램 카운터값이다. 그 다음 값은 인터럽트된 스레드에 대한 현재 스택 프레임의 최상부에 대한 포인터이다. 도시된 실시예에서, 이 값은 스택 프레임(208) 내의 EBP(208a)의 위치를 지정한다. 다음으로, EBP(208)는 스택 프레임(206) 내의 EBP(206a)와 스택 프레임(204) 내의 EBP(204a)의 위치를 차례로 지정한다. 그 후, 상기 EBP는 스택 프레임(202) 내의 EBP(202a)의 위치를 지정한다. 스택 프레임(202-208) 내에는 호출 루틴 복귀 어드레스를 식별하는 EIP(202a-208a)가 존재한다. 루틴은 이들 어드레스로부터 식별될 수 있다. 따라서, 루틴은 스택을 통해 상방향 또는 하방향으로 탐색하여 모든 복귀 어드레스를 수집함으로써 정해될 수 있다.

도 3을 참조하면, 본 발명의 바람직한 실시예에 따른 호출 스택 샘플이 예시되어 있다. 호출 스택 샘플(300)과 같은 소정의 호출 스택 샘플이 호출 스택을 탐지함으로써 얻어진다. 하나의 호출 스택 샘플은, 예를 들어 타이머 인터럽트의 발생과 같은 주기적인 이벤트가 발생할 때마다 얻어진다. 얻어진 이들 호출 스택 샘플은 저장되어 나중에 처리되거나, 또는 프로그램이 계속 실행되는 동안 처리될 수 있다.

도시된 실시예에서, 호출 스택 샘플(300)은 하나의 프로세스 식별자(302)(pid: process identifier)와 하나의 스레드 식별자(thread identifier; tid)(304)를 포함한다. 호출 스택 샘플(300)은 또한 어드레스 1(306)(addr1), 어드레스 2(308)(addr2), ..., 어드레스 N(310)(addrN)을 포함하고 있다. 본 실시예에서, 어드레스 1(306)은 인터럽트 발생 시의 프로그램 카운터의 값이다. 어드레스 1은 인터럽트된 기능의 범위 내의 어느 위치에서 발생한다. 어드레스 2(308)은 인터럽트된 기능을 호출한 기능 내의 어드레스를 나타낸다. 인텔 프로세서 기반 데이터 처리 시스템의 경우, 어드레스 2는 해당 호출에 대한 복귀 어드레스를 표시하는데, 해당 값을 4만큼 감소시키면 호출 위치(call-site)라고도 알려진 실제 호출 어드레스가 된다. 이것은 도 2의 EIP(208b)와 대응한다. 어드레스 N(310)은 호출 스택의 최상부[EIP(202b)]이다. 호출 스택 상태가 도 2에 도시되어 있는 스레드를 타이머 인터럽트가 인터럽트한 경우, 복귀되는 호출 스택 샘플은 인터럽트된 스레드에 대한 프로세스 식별자, 인터럽트된 스레드에 대한 스레드 식별자, 인터럽트된 스레드에 대한 프로세스 프로그램 카운터 값(도 2에는 도시되지 않음), EIP(208b), EIP(206b), EIP(204b) 및 EIP(202b)로 이루어져 있다. 도 3의 경우, 프로그램 카운터값 = 어드레스 1, EIP(208b) = 어드레스 2, EIP(206b) = 어드레스 3, EIP(204b) = 어드레스 4 및 EIP(202b) = 어드레스 5이다.

도 4a를 참조하면, 각 기능 진입부/출구부 지점에서 호출 스택의 상태와 함께 프로그램 실행 시퀀스의 다이어그램이 제공되어 있다. 도 4에는 일정한 시간 간격으로 발생하는 진입부 및 출구부를 예시하고 있으나, 이는 단지 설명 편의를 위한 것이다. 각 기능(도 4에서는 A, B, C 및 X)이 진입부/출구부 이벤트 후크(entry/exit event hook)로 구현되면, 각 기능 이하에서 소비된 시간의 전체 계정이 쉽게 얻어질 수 있다. 도 4a에는, 시점 0에서 실행 스레드가 루틴 C에 있다는 점에 유의하여야 한다. 시점 0에서 호출 스택은 C이다. 시점 1에서 루틴 C는 루틴 A를 호출하여 호출 스택은 CA가 되고, 나머지도 이와 동일한 방식으로 이루어진다. 계정 기법 및 데이터 구조가 본 출원인에게 함께 양도되었으며, 발명의 명칭이 SYSTEM AND METHOD FOR PROVIDING TRACE INFORMATION REDUCTION이고, 1997년 12월 12일자로 출원된 미국 특허 출원 번호 제 08/989,725호에 상세히 기술되어 있는데, 이 출원에서는 전체 계정이 상기 형태의 구현예를 갖는 것을 가능하게 한다. 유감스럽게도, 상기 형태의 구현예는 가격이 비싸, 불균형을 일으킬 수 있으며, 소정의 경우에는 적용하기 어렵다. 본 발명은 진입부/출구부 후크가 프로그램 호출 스택을 샘플링하여 생성되도록 성능 상의 불균형 (및 기타 복잡성)을 제거하는 것을 목적으로 하고 있다.

동일한 프로그램이 실행되지만 일정하게 샘플링되는 도 4b를 참조한다(본 실시예에서, 인터럽트는 2개의 시간 스텝 값에 해당하는 주파수에서 발생한다). 각 샘플은 인터럽트된 스레드 호출 스택의 스냅샷(snapshot)을 포함한다. 모든 호출 스택 조합이 이러한 기법을 사용하는 것은 아니라는 것을 알 수 있다(루틴 X는 도 4b의 호출 스택 샘플에 전혀 나타나 있지 않음에 유의하여야 한다). 이것은 수용 가능한 샘플링 제한 사항이다. 상기 아이디어는 적절한 샘플링 속도(예를 들어, 매초 30 - 100번)로 대부분의 시간이 소비되는 호출 스택이 식별되는 것을 의미한다. 상기 아이디어는 시간을 거의 소비하지 않는 호출 스택 조합이 존재하더라도, 이들 호출 스택 조합의 일부가 남아 있는지의 여부에 전혀 무관하다.

상세한 추적과 연관된 성능 데이터 및 도 4의 호출 스택 데이터는 트리 포맷으로 표시될 수 있다. 또한, 도 4b에 예시된 샘플링된 호출 스택 데이터 등으로부터 얻어진 성능 데이터를 동일한 종류의 트리 구조로 맵핑하는 것이 가능하다.

도 5를 참조하면, 본 발명의 실시예에 따른 호출 스택의 샘플링으로부터 발생된 트리 구조의 다이어그램이 도시되어 있다. 도 5는 트리 구조(500)를 예시하고 있는데, 이 트리 구조(500)의 각 노드는 기능 진입부 지점을 나타낸다. 또한,

트리 구조(500)의 각 노드에는 다수의 통계값이 기록된다. 도시된 실시예에서, 각 노드(502 - 508)는 어드레스(addr), 기준 시간(BASE), 누적 시간(CUM) 및 모와 자 포인터(parent and children pointer)를 포함한다. 어드레스는 기능 진입부 지점을 나타낸다. 기준 시간은 해당 기능을 수행하는 해당 스레드에 의해 직접 소비된 시간량을 나타낸다. 누적 시간은 해당 기능 및 호출 스택 상의 해당 기능 이하에 있는 모든 기능을 실행하는 해당 스레드에 의해 직접 소비된 시간량을 나타낸다. 도시된 실시예에서, 각 노드에 대한 포인터가 포함된다. 하나의 포인터는 해당 노드의 모에 대한 포인터인 모 포인터이다. 각 노드는 또한 해당 노드의 각 자(children)에 대한 포인터를 포함한다.

당업자는 트리 구조(500)가 다양한 방법으로 구현될 수 있다는 점과 도시된 실시예가 아닌 노드에서 여러 가지 상이한 형태의 통계값을 보유할 수 있다는 점을 이해할 수 있다. 또한, 다른 포인터들이 노드 내에 저장되어 추가적인 후속 분석을 행하는데 도움을 줄 수 있다. 또한, 예를 들어 루틴의 이름과 같은 루틴의 속성에 대한 테이블과 같은 기타 구조적인 요소들이 노드 내에 저장될 수 있다. 나아가, 트리 구조 내의 노드는 또한 기능 + 오프셋과 연관을 가질 수 있다. 모든 복귀 어드레스에서 역탐색함으로써 호출 스택이 형성된다. 이들 복귀 어드레스는 특정 오프셋에서 상기 기능들의 본체 내에서 도출된다. 이러한 정보는 동일 기능에 대한 구별 가능한 호출과 호출 사이의 계정 구분을 가능하게 한다. 즉, 기능 X가 기능 A에 대해 2가지 상이한 호출을 가지면, 이들 호출과 관련된 시간은 개별적으로 계산될 수 있다.

도 6을 참조하면, 본 발명의 바람직한 실시예에 따라 호출 스택 트리 구조를 생성하는 프로세스의 플로우차트가 도시되어 있다. 이러한 프로세스는 호출 스택 샘플을 판독함으로써 시작된다[단계(600)]. 이 단계는 타이머 인터럽트가 발생할 때 어떤 루틴이 실행되고 있는지를 결정하기 위하여 호출 스택을 샘플링한다. 진입부와 출구부 후크(exit hook)를 필요로 하는 시스템과는 달리, 본 발명은 호출 스택을 사용하므로 성능 및 루틴에서의 소비 시간에 영향을 미치는 추가 인스트럭션을 프로그램에 추가할 필요가 없다. 다음에, 상기 프로세스/스레드에 대한 트리 구조(pid, tid)의 위치를 찾아낸다[단계(602)]. 다음에, 포인터(PTR)는 $PTR = \text{루트}(pid, tid)$ 로 설정하여 상기 트리 구조의 루트(root)가 되도록 설정된다[단계(604)]. 인덱스(index)는 호출 스택 샘플 내의 진입부 수와 같은 N으로 설정된다[단계(606)].

인덱스가 0인지의 여부에 대한 판단이 이루어진다[단계(608)]. 인덱스가 0이면, 프로세스는 처리될 호출 스택 샘플이 남아 있는지의 여부를 판단하는 단계(610)로 진행한다. 추가 호출 스택 샘플이 남아 있는 경우, 프로세스는 또 다른 호출 스택 샘플을 판독하는 단계(600)로 복귀된다. 호출 스택 샘플이 남아 있지 않은 경우에는 프로세스가 종료된다.

한편, 인덱스가 0이 아닌 경우, sample_address로 표시되는 샘플 어드레스가 call stack address[index]로 표시되는 호출 스택 어드레스로 설정된다[단계(612)]. 다음에, PTR의 모든 자에 대해 $PTR.child.addr = sample_address$ 인지에 대한 판단이 이루어진다[단계(614)]. 즉, 이 단계는 샘플 어드레스가 트리 구조 내에서 상기 레벨(level)에서 발견된 적이 있었는지를 판단한다. 어드레스가 트리 구조 내에서 상기 레벨에서 발견된 적이 없는 경우, PTR의 새로운 자가 생성되어 PTR.child.addr이 샘플 어드레스인 sample_address로 설정되고, 이 노드에 대한 변수 PTR.child.BASE는 0으로 설정되며, 이 노드에 대한 변수 PTR.child.CUM도 0으로 설정된다[단계(616)]. 그 후, 변수 PTR.child.CUM을 증가시켜 상기 노드에 대한 누적 시간(cumulative time)을 증가시킨다[단계(618)]. 어드레스가 상기 레벨에서 발견된 경우, 프로세스는 단계(614)로부터 단계(618)로 진행된다.

다음에, 샘플 어드레스(sample_address)가 호출 스택 샘플의 마지막 어드레스(call stack address[1])인지에 대한 판단이 이루어진다[단계(620)]. 샘플 어드레스가 처리 중인 어드레스인 경우, 변수 PTR.child.BASE를 증가시켜 해당 노드에 대한 기준 시간(base time)을 증가시킨다[단계(622)]. 그 후, 포인터(PTR)는 자로 설정되고[단계(624)], 인덱스가 감소[단계(626)]된 후, 프로세스는 단계(608)로 복귀한다. 단계(620)를 다시 참조하면, 샘플 어드레스가 처리 중인 어드레스가 아닌 경우, 프로세스는 단계(624)로 진행한다.

도 6에 예시된 실시예에서, 프로세스는 프로그램이 실행되는 동안 기록된 호출 스택 샘플을 처리하는데 사용된다. 도시된 프로세스는 또한 프로그램이 실행되는 동안 호출 스택 샘플을 동적으로 처리하도록 구현될 수 있다. 예를 들어, 단계(610)는 다음번 타이머 인터럽트가 발생될 때까지 대기하도록 수정된 후 다음번 인터럽트가 발생하면 루프가 단계(610)로 계속 되돌아 갈 수 있다.

샘플링하는 동안 얻어진 상기 어드레스는 기능들을 식별하는데 사용된다. 기능들은 상기 어드레스를 기능 + 오프셋으로 맵핑하여 식별될 수 있다. 도 7을 참조하면, 샘플링하는 동안 얻어진 소정의 어드레스로부터 기능들을 식별하는데 사용되는, 본 발명의 바람직한 실시예에 따른 프로세스의 플로우차트가 도시되어 있다. 프로세스는 호출 스택을 샘플링하는 동안 얻어지는 프로그램 카운터 값을 판독함으로써 시작된다[단계(700)]. 파일의 끝부분에 도달하였는지의 여부에 대한 결정이 이루어진다[단계(702)]. 파일의 끝부분에 도달하지 않은 경우, 전역 맵(global map) 내의 프로그램 카운터 값이 참조된다. 본 실시예에 도시된 전역 맵은 시스템 맵과 복수의 프로세스 심볼로 이루어지는데, 각 프로세스 심볼은 시스템 로더 정보(system loader information), 애플리케이션, 라이브러리 및 시스템 심볼 테이블로부터 생성된다. 프로세스 + 기능 식별자 (및 오프셋)은 프로그램 카운터 값의 참조에 응답하여 전역 맵으로부터 얻어진다[단계(706)]. 그 후, 프로세스는 단계(700)로 복귀한다. 이러한 방식으로, 기능 + 오프셋 쌍은 어드레스로부터 유도될 수 있다.

기능 + 오프셋 쌍 정보는 이하에서 기술되는 보고서(reports)를 발생시키는데 사용될 수 있다. 도 7의 프로세스는 또한 샘플링된 프로그램의 실행 도중에 사용될 수 있다.

도 8을 참조하면, 본 발명의 프로세스를 사용하여 얻어지는 구조화된 프로파일의 다이어그램이 예시되어 있다. 프로파일(800)의 칼럼(802)에는 샘플 번호가 도시되어 있다. 칼럼(804)은 기능 식별이 상이한 샘플 시점에서 호출 스택 내에 존재하는 호출 스택을 도시하고 있다.

도 9를 참조하면, 본 발명의 프로세스를 사용하여 발생하는 기록 다이어그램이 도시되어 있다. 도 9의 루틴에 대한 정보와 함께 기록(900) 내의 각 루틴이 개별적으로 나열되어 있다. 예를 들어, 샘플 칼럼(902)은 샘플 번호를 식별한다. 다음에, 호출 칼럼(904)은 각 루틴이 호출된 횟수를 나열한다. 기준(BASE) 칼럼은 해당 루틴에서 소비된 전체 시간을 포함하고, 누적(CUM) 칼럼(808)은 해당 루틴 및 그 해당 루틴이 호출한 모든 루틴에서 소비된 누적 시간을 포함한다. 명칭 칼럼(910)은 루틴의 명칭을 포함한다.

도 10은 본 발명의 바람직한 실시예에 따라 생성될 수 있는 또 다른 형태의 보고서의 다이어그램이 도시되어 있다. 도 10에 도시된 보고서는 도 9에서 알 수 있는 동일한 정보의 상당 부분을 예시하고 있지만, 그 포맷은 약간 상이하다. 도 9의 경우와 같이, 다이어그램(1000)은 호출, 기준 시간 및 누적 시간에 대한 정보를 포함한다.

도 2 내지 도 10에 예시된 실시예는 데이터 처리 시스템 및 데이터 처리 시스템 내에서 실행 중인 애플리케이션의 구조화된 프로파일링을 예시하고 있다. 이들 도면에 예시된 실시예는 타이머 인터럽트의 발생에 응답하여 스택으로부터 정보를 얻는 것에 관한 것이다. 본 발명의 프로세스는 또한 데이터 처리 시스템 또는 애플리케이션 내에서 발생하는 다른 형태의 이벤트에도 적용될 수 있다. 예를 들어, 본 발명의 또 다른 실시예에 있어서, 기술된 프로세스는 메모리 분석에 적용될 수 있다. 타이머 인터럽트를 사용하는 대신, 페이지 폴트 인터럽트가 신호로 사용되어 스택으로부터 데이터를 수집할 수 있다. 도시된 실시예에서, 데이터 처리 시스템 내에서 메모리에 대한 액세스가 발생할 때마다 페이지 폴트가 발생된다. 페이지 폴트는 또한 처리 자원에 따라 선택적으로 트리거된다. 예를 들어, 페이지 폴트는 새로운 페이지 또는 상이한 페이지가 액세스되거나, 새로운 데이터 페이지가 액세스되거나, 또는 인스트럭션 페이지가 액세스될 때마다 트리거될 수 있다. 따라서, 페이지 폴트는 선택된 형태의 메모리 액세스가 발생할 때마다 초기화될 수 있다. 본 발명에 사용되는 선택된 형태의 메모리 액세스는 모든 메모리 액세스에서부터 예를 들어 새로운 페이지 액세스 또는 상이한 페이지 액세스와 같은 선택된 액세스까지의 범위를 가질 수 있다. 페이지 폴트를 강제로 발생시키는 것은 시스템 내의 물리적인 페이지 프레임의 수를 더 작은 수의 물리적 페이지 프레임으로 감소시키는 것과 같은 여러 가지 알려진 메커니즘

을 통해 이루어질 수 있다. 예를 들어, 페이지 프레임의 수는 데이터 처리 시스템의 아키텍처가 필요로 하는 최소값 이상인 소정의 수까지 감소될 수 있다. 페이지 폴트가 발생할 경우, 페이지 폴트 어드레스 및 페이지 폴트 인스트럭션 어드레스가 얻어질 수 있다. 이러한 정보로부터, 상기된 바와 같은 본 발명의 프로세스 및 이하에서 기술되는 추가 프로세스가 메모리 사용을 분석하는데 사용될 수 있다.

특히, 도 2에 도시된 스택과 같은 호출 스택이 검사되고, 도 5의 트리 구조(500)와 같은 트리 구조가 도 6에 기술된 프로세스를 사용하여 생성된다. 기준 시간 및 누적 시간 대신, 메모리 액세스의 기준 횟수 및 액세스의 누적 횟수가 메모리 분석시 추적된다. 그 결과, BASE는 메모리 액세스가 특정 기능을 실행하는 특정 스레드에 의해 직접 발생하는 횟수이고, CUM은 특정 기능 및 호출 스택 상의 기능 및 그 이하의 모든 기능을 실행하는 스레드에 의한 메모리 액세스의 누적 횟수이다.

도 11을 참조하면, 본 발명의 바람직한 실시예에 따른 메모리 액세스 분석용 프로세스의 플로우차트가 도시되어 있다. 프로세스는 페이지 폴트 어드레스 및 페이지 폴트 인스트럭션 어드레스를 수집함으로써 시작되고[단계(1100)], 정보는 호출 스택으로부터 수집된다[단계(1102)]. 다음에, 어드레스가 인스트럭션 어드레스인지 데이터 어드레스인지를 결정함으로써 페이지 폴트가 인스트럭션 폴트인지 데이터 폴트인지의 여부에 대한 결정이 이루어진다[단계(1104)]. 페이지 폴트 어드레스 및 페이지 폴트 인스트럭션 어드레스가 동일하면, 페이지 폴트는 인스트럭션 폴트이다. 그렇지 않으면, 페이지 폴트는 데이터 폴트이다. 어드레스가 인스트럭션 어드레스이면, 프로세스는 종료된다.

폴트가 데이터 폴트이면, 프로세스는 데이터 어드레스를 모듈 어드레스, 스택 어드레스 또는 힙 어드레스(heap address) 중 어느 하나로 분류한다[단계(1106)]. 어드레스가 모듈 어드레스이면, 소정의 모듈 식별 정보가 모듈의 시작으로부터 오프셋과 함께 수집되고[단계(1108)], 그 후 프로세스가 종료된다. 데이터 어드레스가 스택 어드레스로 분류되면, 스택 프레임 내로부터 오프셋이 수집되고[단계(1110)], 그 후 프로세스가 종료된다. 데이터 어드레스가 힙 어드레스로 분류되면, 힙의 시작으로부터의 오프셋이 수집되고[단계(1112)], 그 후 프로세스가 종료된다.

도 12를 참조하면, 본 발명의 바람직한 실시예에 따라 페이지 폴트에 응답하여 얻어진 어드레스로부터의 기능을 식별하기 위한 프로세스의 플로우차트가 도시되어 있다. 도 12에 도시된 플로우차트는 사후 처리 방식으로 실행될 수 있는 프로세스이다. 이들 프로세스는 도 7에 예시된 프로세스와 함께 사용되어 메모리 분석용 정보를 얻을 수 있다. 도 12의 프로세스는 여러 가지 형태의 데이터 어드레스를 분석하는데 사용되고, 도 7의 프로세스는 페이지 폴트의 발생에 응답하여 호출 스택으로부터 정보를 얻기 위한 메모리 분석에 사용된다.

도 12의 프로세스는 파일로부터 데이터 정보를 얻기 위해 소정의 값을 판독함으로써 시작된다[단계(1200)]. 파일의 끝부분에 도달하였는지의 여부에 대한 결정이 이루어진다[단계(1202)]. 파일의 끝부분에 도달한 경우, 프로세스는 종료된다. 그렇지 않은 경우, 데이터 형태를 식별하기 위한 결정이 이루어진다[단계(1204)]. 데이터가 스택으로부터의 것이면, 심볼 파일로부터 데이터 명칭이 참조되고[단계(1206)], 그 후 프로세스는 단계(1200)로 복귀된다. 데이터가 모듈로부터의 것이면, 심볼 파일로부터 데이터 명칭이 참조되고[단계(1208)], 그 후 프로세스는 단계(1200)로 복귀된다. 데이터가 힙으로부터의 것이라는 식별에 응답하여, 힙 및 오프셋이 보고서에 인쇄 또는 위치되고[단계(1210)], 그 후 프로세스는 단계(1200)로 복귀된다.

메모리 분석시, 도 7 및 도 12에 예시된 프로세스는 사후 처리가 아니라 도 6에 도시된 바와 같이 처리가 이루어지는 동안에 실행될 수 있다. 동시 처리는 통상적으로 도 6의 프로세스가 실행되는 동안 도 7 및 도 12의 프로세스를 실행하는데 드는 시간 비용이 동시 처리를 할 경우의 이익을 능가하지 않는 경우에만 행해진다.

도 13을 참조하면, 본 발명의 프로세스를 사용하여 얻어지는 메모리 프로파일의 다이어그램이 예시되어 있다. 프로파일(1300)은 프로그램 또는 데이터 처리 시스템을 분석하는 동안 수집된 샘플의 일부를 도시하고 있다. 샘플 번호 칼럼(1302)은 샘플 번호를 식별하는데, 이 샘플 번호는 페이지 폴트에 의해 표시되는 메모리 액세스를 나타낸다. 데이터 칼럼(1304)은 페이지 폴트에 응답하여 수집된 데이터를 나타낸다. 프로파일(1300)에서, 샘플 1 및 샘플 3은 인스트럭션 폴트이다. 샘플 2는 모듈로부터의 데이터 폴트이다. 샘플 2에 대한 데이터 칼럼(1304)에서, 0x0005는 모듈 정보이고, 0x000000008은 오프셋이다. 프로파일(1300)의 샘플 4는 스택으로부터의 데이터 폴트이다. 샘플 4에서, 0x00000004는 스택 내의 오프셋이다. 기능 f4()에 대한 정보는 오프셋에 대응하는 기능의 명칭을 식별할 수 있도록 해준다. 프로파일(1300)의 샘플 5는 힙으로부터의 데이터 폴트이다. 데이터 칼럼(1304)에서 0x00000100은 힙 내의 오프셋이다.

도 14를 참조하면, 본 발명의 바람직한 실시예에 따른 인스트럭션 폴트용 샘플 x트리 보고서가 도시되어 있다. x트리 보고서는 도 10에 예시된 보고서와 같은 보고서의 트리 뷰(tree view)이다. x트리 보고서(1400)에서, 라인(1402, 1404, 1406, 1408)은 도 13으로부터의 프로파일(1300) 내의 샘플 1을 나타낸다. 각 라인은 기능의 식별자와 그 기능에 의해 액세스되는 횟수를 포함한다. 예를 들어, 라인(1408)에서 기능은 f3()이고 액세스 횟수는 4회인 반면, 라인(1406)에서 기능의 명칭은 A이고 액세스 횟수는 2회이다. 라인(1410)은 도 13의 프로파일(1300)의 샘플 2를 나타낸다. 도 14에서, 라인(1404)의 X() 및 라인(1410)의 Y()는 라인(1402)의 Main()의 자이다. 라인(1406)에서, a()는 X()의 자이고, 라인(1408)의 f3()은 a()의 자이다.

도 15를 참조하면, 본 발명의 바람직한 실시예에 따른 데이터 폴트용 샘플 x트리 보고서가 도시되어 있다. x트리 보고서(1500)에서, 라인(1502, 1504, 1506, 1508, 1510)은 도 13의 프로파일(1300)로부터의 샘플 4를 나타낸다. 라인(1510)에서, (m)은 변수 var_x가 모듈 변수임을 나타내고, 숫자 4는 상기 모듈 변수가 4회 액세스되었음을 나타낸다. 라인(1512)에서, (s)는 param_y가 (전달된 파라미터 또는 로컬 변수로) 스택에 있었음을 나타내며, 숫자 2는 상기 변수가 2회 액세스되었음을 나타낸다. 라인(1514)에서, (h)는 정보가 힙으로부터의 것이며, 숫자 3으로 표시되는 바와 같이 오프셋 0x000100이 3회 액세스되었음을 나타낸다. 도시된 실시예들에서, 기능은 더 이상 타이머 샘플링에서와 같은 잎 노드(leaf node)가 아니다. 대신, 변수가 잎 노드를 형성한다. 예를 들어, 라인(1510)에서는 기능 f3() 대신 var_x가 잎 노드가 된다.

발명의 효과

따라서, 본 발명은 데이터 처리 시스템 및 데이터 처리 시스템 내에서 실행 중인 애플리케이션의 구조화된 프로파일링을 위한 개선된 방법 및 장치를 제공한다. 이러한 장점은 호출 스택으로부터의 정보를 포함하고 프로파일을 제공하여 단지 잎뿐만 아니라 전체 호출 스택의 샘플링을 반영하는 프로그램 샘플링에 의해 제공된다. 또한, 본 발명은 다른 형태의 스택에도 적용될 수 있다. 예를 들어, 자바(Java) 프로그램의 경우 번역 프로그램(interpreter)이라고 불리우는 서브루틴에서 상당량의 시간이 소비된다. 호출 스택만이 검사된 경우, 프로파일은 많은 유용한 정보를 나타내지 않는다. 번역 프로그램은 또한 자신의 스택 내에 있는 정보[예컨대, 자바 스택 (및 자신의 연결 규약)]를 추적하기 때문에, 프로세스는 번역된 자바 프로그램의 예측으로부터 호출 시퀀스를 얻기 위해 자바 스택을 상방향으로 탐색하는데 사용될 수 있다.

본 발명이 전체 기능을 수행하는 데이터 처리 시스템과 관련하여 기술하였지만, 당업자는 본 발명의 프로세스가 인스트럭션의 컴퓨터 판독 가능 매체의 형태 및 다양한 형태로 배포될 수 있으며, 본 발명이 배포되는데 실제로 사용되는 신호 포함 매체의 특정 형태와는 무관하게 동일하게 적용될 수 있다는 점에 유의하여야 한다. 컴퓨터 판독 가능 매체의 예로는 플로피 디스크, 하드 디스크 드라이브, RAM 및 CD-ROM과 같은 기록 가능형 매체와 디지털 및 아날로그 통신 링크와 같은 전송형 매체를 들 수 있다.

본 발명이 예시 목적으로 기술되었지만, 개시된 형태의 발명만을 망라하거나 이러한 발명에만 제한되는 것은 아니다. 다수의 수정 및 변경이 이루어질 수 있다는 점이 당업자에게 명백히 이해될 수 있다. 예를 들어, 도시된 실시예는 타이머 인터럽트를 사용하고 있지만, 기타 다른 인터럽트가 기술된 샘플링 메커니즘을 트리거하는데 사용될 수 있다. 예시된 실시예는 본 발명의 원리를 가장 잘 설명하기 위해 선택되고 기술되었으며, 이를 실제에 응용하면 당업자는 본 발명의 다양한 실시예가 고려될 수 있는 특정 용도에 적합하도록 여러 가지로 변경될 수 있다는 점을 이해할 수 있다.

(57) 청구의 범위

청구항 1.

프로그램의 성능을 모니터링하는 방법에 있어서,

- a) 선택된 이벤트 발생을 검출하는 단계와;
- b) 상기 선택된 이벤트의 검출에 응답하여 프로그램과 연관된 호출 스택(call stack)을 식별하는 단계와;
- c) 상기 프로그램과 관련하여 현재 실행 중인 각 루틴을 식별하기 위해 호출 스택을 검사하는 단계와;
- d) 각 루틴을 트리 구조의 노드로 표시하는 단계를 포함하는 프로그램 성능 모니터링 방법.

청구항 3.

제1항에 있어서, 상기 실행과 관련된 정보는 기능 명칭 + 오프셋을 포함하는 것인 프로그램 성능 모니터링 방법.

청구항 11.

프로그램의 성능을 모니터링하는 데이터 처리 시스템에 있어서,

- a) 선택된 이벤트 발생을 검출하는 검출 수단과;
- b) 상기 선택된 이벤트의 검출에 응답하여 프로그램과 연관된 호출 스택(call stack)을 식별하는 식별 수단과;
- c) 상기 프로그램과 관련하여 현재 실행 중인 각 루틴을 식별하기 위해 호출 스택을 검사하는 검사 수단과;
- d) 각 루틴을 트리 구조의 노드로 표시하는 표시 수단을 포함하는 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 12.

제11항에 있어서, 각 루틴의 실행과 관련된 정보를 상기 루틴을 표시하는데 사용되는 노드와 연관시키는 연관 수단을 더 포함하는 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 13.

제11항에 있어서, 현재 실행 중인 각 루틴을 식별하는 보고서를 생성하는 생성 수단을 더 포함하는 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 14.

제13항에 있어서, 상기 생성 수단은 호출 스택이 검사될 때마다 활성화되는 것인 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 15.

제13항에 있어서, 상기 생성 수단은 프로그램 실행이 종료된 후에 활성화되는 것인 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 16.

제13항에 있어서, 상기 보고서는 트리 구조 포맷으로 되어 있는 것인 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 17.

제11항에 있어서, 상기 선택된 이벤트는 이벤트 타이머(event timer)인 것인 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 18.

제11항에 있어서, 상기 검사 수단은,

- a) 상기 호출 스택—여기서, 호출 스택은 주기적으로 발생하는 이벤트가 발생할 때 실행 중인 루틴의 목록임—을 판독하는 판독 수단과;
- b) 선택된 루틴이 상기 트리 구조 내에서 발견되는지의 여부를 결정하는 결정 수단과;
- c) 상기 선택된 루틴이 트리 구조에 존재하지 않는다는 결정에 응답하여 어드레스용으로 사용되는 새로운 노드—여기서, 새로운 노드는 또 다른 어드레스를 표시하는 모 노드(parent node)로부터 파생된 자 노드(children node)임—를 상기 트리 구조에 추가하는 추가 수단을 포함하는 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 19.

제17항에 있어서, 상기 루틴의 목록은 복수의 어드레스이며, 상기 복수의 어드레스 내의 각 어드레스는 진입부 지점(entry point) 또는 루틴 내의 다른 오프셋을 식별하는 것인 프로그램 성능 모니터링용 데이터 처리 시스템.

청구항 20.

프로그램의 성능을 모니터링하는 데이터 처리 시스템에 사용되는 컴퓨터 프로그램을 포함하는 컴퓨터 판독 가능 기록 매체에 있어서, 상기 컴퓨터 프로그램은,

- a) 주기적으로 발생하는 이벤트를 검출하는 제1 인스트럭션과;
- b) 상기 주기적으로 발생하는 이벤트의 검출에 응답하여 프로그램과 연관된 호출 스택(call stack)을 식별하는 제2 인스트럭션과;

- c) 상기 프로그램과 관련하여 현재 실행 중인 각 루틴을 식별하기 위해 호출 스택을 검사하는 제3 인스트럭션과;
- d) 각 루틴을 트리 구조의 노드로 표시하는 제4 인스트럭션을 포함하는 것인 기록 매체.

청구항 21.

제20항에 있어서, 상기 컴퓨터 프로그램은 각 루틴의 실행과 관련된 정보를 상기 루틴을 표시하는데 사용되는 노드와 연관시키는 제5 인스트럭션을 더 포함하는 것인 기록 매체.

청구항 22.

제20항에 있어서, 상기 컴퓨터 프로그램은 현재 실행 중인 각 루틴을 식별하는 보고서를 생성하는 제5 인스트럭션을 더 포함하는 것인 기록 매체.

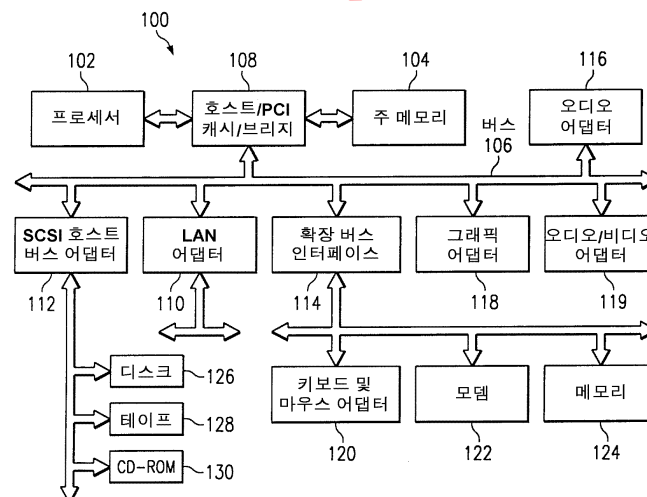
청구항 23.

제20항에 있어서, 상기 호출 스택을 검사하는 제3 인스트럭션은,

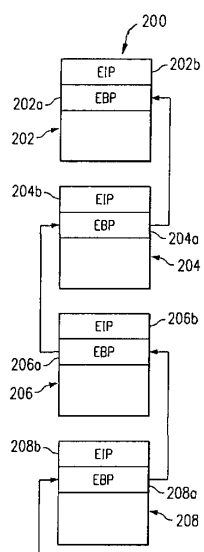
- a) 상기 호출 스택—여기서, 호출 스택은 주기적으로 발생하는 이벤트가 발생할 때 실행 중인 루틴의 목록임—을 판독하는 인스트럭션과;
- b) 선택된 루틴이 상기 트리 구조 내에서 발견되는지의 여부를 결정하는 인스트럭션과;
- c) 상기 선택된 루틴이 상기 트리 구조에 존재하지 않는다는 결정에 응답하여 어드레스용으로 사용되는 새로운 노드—여기서, 새로운 노드는 또 다른 어드레스를 표시하는 모 노드로부터 파생된 자 노드임—을 상기 트리 구조에 추가하는 인스트럭션을 포함하는 것인 기록 매체.

도면

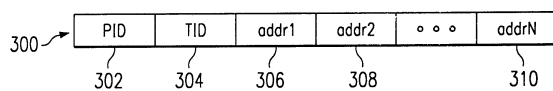
도면 1



도면 2



도면 3



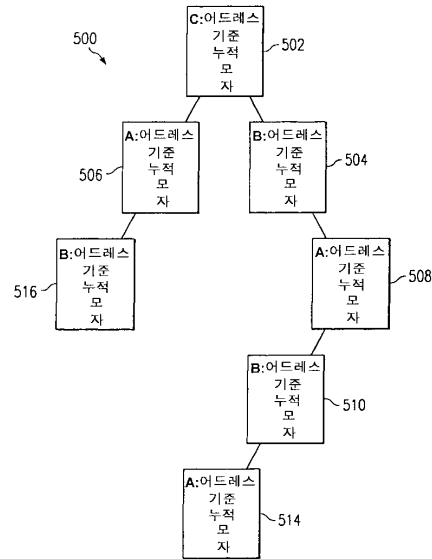
도면 4a

타임스탬프	이벤트	이벤트 후의 호출 스택
0	C로 들어감	C
1	A로 들어감	CA
2	B로 들어감	CAB
3	B로 부터 복귀	CA
4	B로 들어감	CAB
5	B로 들어감	CABB
6	B로 부터 복귀	CAB
7	B로 부터 복귀	CA
8	A로 부터 복귀	C
9	B로 들어감	CB
10	A로 들어감	CBA
11	B로 들어감	CBAB
12	A로 들어감	CBABA
13	A로 부터 복귀	CBAB
14	B로 부터 복귀	CBA
15	X로 들어감	CBAX
16	X로 부터 복귀	CBA
17	A로 부터 복귀	CB
18	B로 부터 복귀	C
19	C로 부터 복귀	

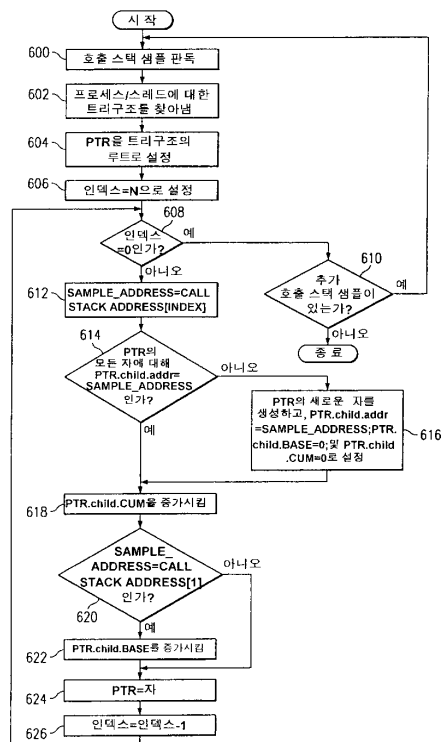
도면 4b

샘플	호출 스택 @ 샘플
1	C
2	CAB
3	CAB
4	CAB
5	C
6	CBA
7	CBABA
8	CBA
9	CBA
10	C

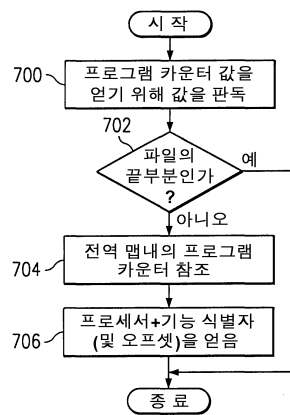
도면 5



도면 6



도면 7



도면 8

802 샘플 번호	804 호출 스택
1	MAIN X() a() f3()
2	MAIN Y() a() f4()
3	MAIN X() b() a() f3()
4	MAIN X() Y() a() f0()

도면 9

902 샘플	904 호출	906 기준	908 누적	910 명칭
1	1	0	19	pt_plt.tid
2	1	3	19	C
3	3	7	14	A
4	5	8	13	B
5	1	1	1	X

도면 10

1000 ↗

ARCFLOW OUTPUT

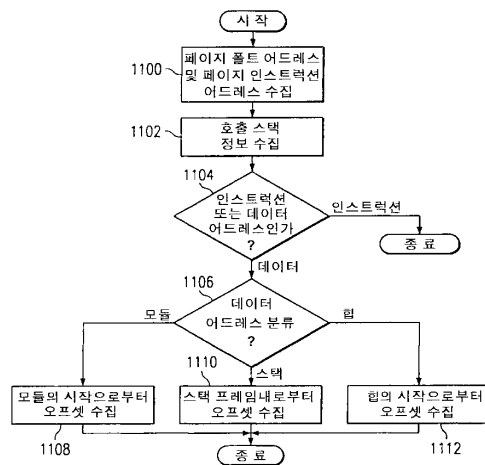
BASE - TIME/INSTRUCTIONS DIRECTLY IN FUNCTION
 CUM - TIME/INSTRUCTIONS DIRECTLY AND INDIRECTLY IN FUNCTION

ARCFLOW INVARIANTS:

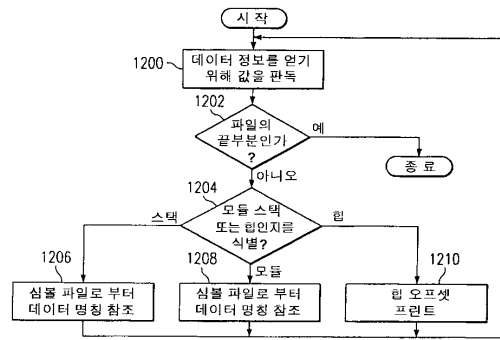
- 1) SUM (PARENT (CALLS)) = SELF (CALLS)
- 2) SUM (PARENT (BASE)) = SELF (BASE)
- 3) SUM (PARENT (CUM)) = SELF (CUM)
- 4) SUM (CHILD (CUM)) = SELF (CUM) - SELF (BASE)

소스	호출	기준	누적	기능
자신	1	0	19	[0] pL_pidtid
자	1	3	19	C
모	1	3	19	pL_pidtid
자신	1	3	19	[1] C
자	1	2	9	B
자	1	3	7	A
모	1	3	7	C
모	1	3	7	B
r 모	1	1	1	B
자신	3	7	14	[2] A
			15	
자	3	5	7	B
자	1	1	1	X
모	2	3	4	A
r 모	1	2	3	A
모	1	2	9	C
자신	5	8	13	[3] B
			17	
자	1	3	7	A
자	1	1	1	A
자	1	1	1	B
모	1	1	1	A
자신	1	1	1	[4] X

도면 11



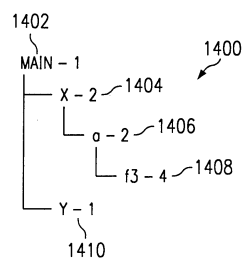
도면 12



도면 13

1302	1304
1	MAIN X() a() f3() I
2	MAIN X() a() f3() M 0X0005 0X00000008
3	MAIN Y() I
4	MAIN Y() a() f4() \$ 0X00000004
5	MAIN X() b() a() f3() # 0X00000100

도면 14



도면 15

