(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2009년08월04일
(11) 등록번호 10-0910821
(24) 등록일자 2009년07월29일

(51) Int. Cl.
G06F 9/46 (2006.01) *G06F 15/173* (2006.01)
(21) 출원번호 10-2007-7009206
(22) 출원일자 2005년10월13일
심사청구일자 2008년06월27일
(85) 번역문제출일자 2007년04월23일
(65) 공개번호 10-2007-0064646
(43) 공개일자 2007년06월21일
(86) 국제출원번호 PCT/EP2005/055240
(87) 국제공개번호 WO 2006/045704
국제공개일자 2006년05월04일
(30) 우선권주장
10/974,514 2004년10월27일 미국(US)
(56) 선행기술조사문헌
2001 OLS'01, Paul E. McKenney외 5인 공저, "read copy update"
전체 청구항 수 : 총 10 항

(73) 특허권자
인터내셔널 비지네스 머신즈 코포레이션
미국 10504 뉴욕주 아몬크 뉴오차드 로드
(72) 발명자
맥켄니 폴
미국 오레곤주 97006 베버톤 노쓰웨스트 알비온 코트 1975
러셀 폴
오스트렐리아 뉴 사우쓰 웨일즈 퀸베이얀 아담스 스트리트 1/7
사마 디팽커
인도 560038 카나타케 방갈로어 인디라나져 세컨드 스테이지크로스 이. 알.제이. 가든 아파트먼트 17쓰 3030이
(74) 대리인
김태홍, 송승필

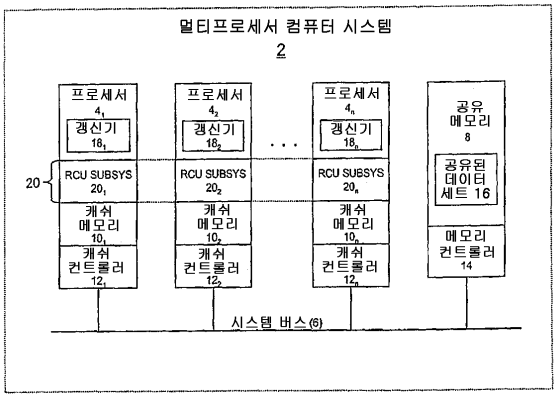
심사관 : 황승희

(54) 아토믹 명령어없이 많은 수의 프로세서들을 무리없이유연하게 처리하는 판독-복사 갱신 유예 기간 검출 방법

(57) 요 약

공유된 데이터 엘리먼트의 제거를, 상기 데이터 엘리먼트에 대한 기존 참조가 제거될때까지 연기시킬 것을 필요로하는 기타 프로세싱 환경에서 아토믹 명령어없이 유예 기간을 탐지하는 방법, 시스템 및 컴퓨터 프로그램 제품이 개시된다. 유예 기간의 탐지 방법은, 공유된 데이터 엘리먼트에 대한 액세스를 공유하는 프로세싱 실체들 사이에서 순환될 토큰을 확립하는 단계를 포함한다. 유예 기간은, 토큰이 프로세싱 실체들 사이를 왕복할 때마다 경과하는 것으로 결정될 수 있다. 분배된 인디케이터는 프로세싱 실체들이 토큰 프로세싱에 참여하기 이전에 각각의 프로세싱 실체에서 프로세스된다. 토큰 프로세싱은 분배된 인디케이터에 의해 허가된 경우에만 프로세싱 실체에서 수행된다. 이러한 방식으로, 분배된 인디케이터가 불필요한 토큰 프로세싱을 허가하지 않는 경우, 불필요한 토큰 프로세싱이 방지될 수 있다.

대 표 도 - 도4



특허청구의 범위

청구항 1

공유된 데이터 엘리먼트의 제거를, 상기 공유된 데이터 엘리먼트에 대한 기존의 참조가 제거될 때까지 연기하기 위한 유예 기간(grace period) 탐지 방법에 있어서,

상기 데이터 엘리먼트에 대한 액세스를 공유하는 프로세싱 실체들 사이에서 순환될 토큰을 확립하는 단계;

상기 토큰이 상기 프로세싱 실체들 사이를 왕복(make a round trip)할 때 상기 유예 기간이 경과했다고 결정하는 단계;

상기 데이터 엘리먼트에 대해, 또는 상기 프로세싱 실체들에 의해 공유된 다른 데이터 엘리먼트에 대해, 제거 프로세싱을 행할 필요가 있는지의 여부를 가리키는 분배된 인디케이터(distributed indicator)를, 상기 프로세싱 실체들 각각과 연관시키는 단계;

상기 프로세싱 실체가 토큰 프로세싱에 참여하기 이전에, 상기 프로세싱 실체들 각각에서 상기 분배된 인디케이터를 프로세싱하는 단계; 및

상기 분배된 인디케이터에 의해 허가(warrant)된 경우에만 상기 프로세싱 실체에서 토큰 프로세싱을 수행하는 단계를 포함하고,

이렇게하여, 상기 분배된 인디케이터가 불필요한 토큰 프로세싱을 허가하지 않는 경우에는 불필요한 토큰 프로세싱이 방지될 수 있는 것인, 유예 기간 탐지 방법.

청구항 2

제1항에 있어서, 상기 분배된 인디케이터들 각각은, 상기 프로세싱 실체들 중의 하나와 연관된 캐쉬 메모리 내에 로컬 변수로서 저장되는 것인, 유예 기간 탐지 방법.

청구항 3

제1항에 있어서, 분배된 인디케이터는, 상기 프로세싱 실체들에 의해 공유된 데이터 엘리먼트를 제거하라는 보류중인 요구를 갖거나, 상기 프로세싱 실체들에 의해 공유된 데이터 엘리먼트를 제거하라는 복수의 보류중인 요구를 갖는 복수의 상기 프로세싱 실체들을 표현하는 것인, 유예 기간 탐지 방법.

청구항 4

제1항 내지 제3항 중 어느 한 항에 있어서, 상기 프로세싱 실체들 중 하나에서의 상기 분배된 인디케이터들 중 하나에 가해진 변경은, 상기 프로세싱 실체들 중 다른 것들에 전파되는 것인, 유예 기간 탐지 방법.

청구항 5

하나 이상의 프로세서, 메모리, 및 상기 하나 이상의 프로세서와 메모리 사이의 통신 경로를 갖는 데이터 프로세싱 시스템으로서, 상기 프로세서들 중 한 프로세서에 의한 공유된 데이터 엘리먼트의 제거를, 상기 프로세서들 중 다른 프로세서에 의해 유지된 공유된 데이터 엘리먼트에 대한 기존의 참조가 제거될 때까지 연기시키기 위한 유예 기간을 탐지하도록 적합화된, 상기 데이터 프로세싱 시스템에 있어서,

상기 프로세서들 사이에서 순환되는 토큰(token);

상기 프로세서들 각각과 연관된 토큰 머니퓰레이터(token manipulator)로서, 상기 토큰을 순환시키고, 상기 프로세서들 사이를 왕복하는 상기 토큰에 의해 상기 유예 기간이 경과되었는지의 여부를 결정하도록 적합화된 상기 토큰 머니퓰레이터;

상기 프로세서들 각각과 연관되어 있는 분배된 인디케이터(distributed indicator)로서, 상기 데이터 엘리먼트에 대해 또는 상기 프로세싱 실체에 의해 공유된 다른 데이터 엘리먼트에 대해, 제거 프로세싱을 행할 필요가 있는지의 여부를 가리키는 상기 분배된 인디케이터; 및

상기 토큰 머니퓰레이터에 의한 토큰 프로세싱 이전에 상기 분배된 인디케이터를 프로세스하도록 적합화된 상기 프로세서들 각각과 연관되어 있는, 분배된 인디케이터 핸들링 메커니즘을 포함하고,

상기 분배된 인디케이터 핸들링 메커니즘은, 상기 분배된 인디케이터에 의해 허가된 경우에만 상기 토큰 머니퓰레이터가 상기 프로세서에서 토큰 프로세싱을 수행하는 것을 허용하도록 더 적합화되고;

이렇게하여 상기 분배된 인디케이터가 불필요한 토큰 프로세싱을 허가하지 않는 경우, 불필요한 토큰 프로세싱이 방지될 수 있는 것인, 데이터 프로세싱 시스템.

청구항 6

공유된 데이터 엘리먼트의 제거를, 상기 공유된 데이터 엘리먼트에 대한 기존의 참조가 제거될 때까지 연기하기 위한 유예 기간을 탐지하기 위한 컴퓨터 프로그램을 기록한 컴퓨터 판독가능한 저장 매체로서,

상기 데이터 엘리먼트에 대한 액세스를 공유하는 프로세싱 실체들 사이에서 순환될 토큰을 확립하고;

상기 토큰이 상기 프로세싱 실체들 사이를 왕복할 때 상기 유예 기간이 경과했다고 결정하고;

상기 데이터 엘리먼트에 대해, 또는 상기 프로세싱 실체에 의해 공유된 다른 데이터 엘리먼트에 대해, 제거 프로세싱을 행할 필요가 있는지의 여부를 가리키는 분배된 인디케이터를 상기 프로세싱 실체들 각각과 연관시키고;

상기 프로세싱 실체들이 토큰 프로세싱에 참여하기 이전에 상기 프로세싱 실체들 각각에서 상기 분배된 인디케이터를 프로세싱하고; 그리고

상기 분배된 인디케이터에 의해 허가된 경우에만 상기 프로세싱 실체에서 토큰 프로세싱을 수행하게끔 동작하도록,

데이터 프로세싱 플랫폼을 프로그래밍하기 위한 상기 데이터 저장 매체에 기록된 수단을 포함하고,

이렇게하여, 상기 분배된 인디케이터가 불필요한 토큰 프로세싱을 허가하지 않는 경우 불필요한 토큰 프로세싱이 방지될 수 있는 것인, 컴퓨터 판독가능한 저장 매체.

청구항 7

보류중인 콜백이 실행될 수 있는 때를 결정하기 위해 판독-복사 갱신 서브시스템에서 유예 기간을 탐지하기 위한 방법에 있어서,

데이터 엘리먼트에 대한 액세스를 공유하는 한 세트의 프로세서들 각각에서 정적 상태 카운터(quiescent state counter)를 구현하는 단계;

상기 정적 상태 카운터에 의해 유지된 카운트 값들에서 불연속 값으로서 유예 기간 토큰을 확립하는 단계;

상기 토큰이 상기 프로세서들 중 하나에 복귀하기 위해 상기 한 세트의 프로세서들 사이를 왕복할 때 상기 프로세서들 중 하나에서 상기 유예 기간이 경과했다고 결정하는 단계;

상기 콜백을 프로세스할 필요가 있는지의 여부를 가리키는 분배된 콜백 인디케이터를, 상기 프로세서들 각각과 연관된 캐쉬 메모리에 로컬 변수로서 상기 프로세서의 각각과 연관시키는 단계;

상기 프로세싱 실체들이 토큰 프로세싱에 참여하기 이전에 상기 프로세싱 실체들 각각에서 상기 분배된 콜백 인디케이터를 프로세싱하는 단계; 및

상기 분배된 콜백 인디케이터에 의해 허가된 경우에만 상기 프로세서에서 토큰 프로세싱을 수행하는 단계를 포함하고,

이렇게하여, 상기 분배된 콜백 인디케이터가 불필요한 토큰 프로세싱을 허가하지 않는 경우 불필요한 토큰 프로세싱이 방지될 수 있는 것인, 유예 기간 탐지 방법.

청구항 8

공유된 데이터 엘리먼트의 제거를, 상기 공유된 데이터 엘리먼트에 대한 기존의 참조가 제거될 때까지 연기하기 위한 유예 기간 탐지 방법에 있어서,

상기 데이터 엘리먼트에 대한 액세스를 공유하는 프로세싱 실체들 사이에서 순환될 토큰을 확립하는 단계;

상기 토큰이 상기 프로세싱 실체들 사이를 왕복할 때 상기 유예 기간이 경과했다고 결정하는 단계;

상기 프로세싱 실체에 의해 공유된 데이터 엘리먼트를 제거하라는 보류중인 요구를 갖는 복수의 상기 프로세싱 실체들을 표현하는 분배된 인디케이터를, 상기 프로세싱 실체들 각각과 연관시키는 단계;

상기 프로세싱 실체들이 토큰 프로세싱에 참여하기 이전에 상기 프로세싱 실체들 각각에서 상기 분배된 인디케이터를 프로세싱하는 단계; 및

상기 분배된 인디케이터에 의해 허가된 경우에만 상기 프로세싱 실체에서 토큰 프로세싱을 수행하는 단계를 포함하고,

상기 분배된 인디케이터를 프로세싱하는 단계는, 상기 프로세싱 실체들 중 하나의 이웃 프로세싱 실체에서의 상기 분배된 인디케이터의 값과, 상기 값의 수정에 따라 상기 분배된 인디케이터를 수정하는 단계를 포함하고,

상기 인디케이터의 값의 수정은, (1) 현재 유예 기간과 연관된 제거 요구가 있는지의 여부 및 이전 유예 기간과 연관된 어떠한 제거 요구도 없었는지의 여부에 따라, 상기 분배된 인디케이터가 증가되거나, (2) 현재 유예 기간과 연관된 어떠한 제거 요구도 없는지의 여부 및 이전 유예 기간과 연관된 제거 요구가 있었는지의 여부에 따라, 상기 분배된 인디케이터가 감소되거나, (3) 현재 및 이전 유예 기간 모두에 대한 제거 요구가 있거나, 또는 현재 및 이전 유예 기간 모두에 대한 어떠한 제거 요구도 없는지의 여부에 따라, 상기 분배된 인디케이터가 동일하게 유지되는 방식으로 이루어지며,

이렇게하여 상기 분배된 인디케이터가 불필요한 토큰 프로세싱을 허가하지 않는 경우 불필요한 토큰 프로세싱이 방지될 수 있는 것인, 유예 기간 탐지 방법.

청구항 9

공유된 데이터 엘리먼트의 제거를, 상기 공유된 데이터 엘리먼트에 대한 기존의 참조가 제거될 때까지 지연하기 위한 유예 기간 탐지 방법에 있어서,

상기 데이터 엘리먼트에 대한 액세스를 공유하는 프로세싱 실체들 사이에서 순환될 토큰을 확립하는 단계;

상기 토큰이 상기 프로세싱 실체들 사이를 왕복할 때 상기 유예 기간이 경과했다고 결정하는 단계;

상기 프로세싱 실체들에 의해 공유된 데이터 엘리먼트를 제거하라는 보류중인 요구의 총 갯수를 표현하는 분배된 인디케이터를, 상기 프로세싱 실체들 각각과 연관시키는 단계;

상기 프로세싱 실체들이 토큰 프로세싱에 참여하기 이전에 상기 프로세싱 실체들 각각에서 상기 분배된 인디케이터를 프로세싱하는 단계; 및

상기 분배된 인디케이터에 의해 허가된 경우에만 상기 프로세싱 실체에서 토큰 프로세싱을 수행하는 단계를 포함하고,

상기 분배된 인디케이터를 프로세싱하는 단계는, 상기 프로세싱 실체들 중 하나의 이웃 프로세싱 실체에서 상기 분배된 인디케이터의 값과 현재 유예 기간 동안 추가된 제거 요구의 갯수와 이전 유예 기간 동안 프로세스된 제거 요구의 갯수간의 차이에 따른 상기 값의 수정에 따라 상기 분배된 인디케이터를 수정하는 단계를 포함하고,

이렇게하여 상기 분배된 인디케이터가 불필요한 토큰 프로세싱을 허가하지 않는 경우 불필요한 토큰 프로세싱이 방지될 수 있는 것인, 유예 기간 탐지 방법.

청구항 10

공유된 데이터 엘리먼트의 제거를, 상기 공유된 데이터 엘리먼트에 대한 기존의 참조가 제거될 때까지 지연하기 위한 유예 기간 탐지 방법에 있어서,

상기 데이터 엘리먼트에 대한 액세스를 공유하는 프로세싱 실체들 사이에서 순환될 토큰을 확립하는 단계;

상기 토큰이 상기 프로세싱 실체들 사이를 왕복할 때 상기 유예 기간이 경과했다고 결정하는 단계;

상기 프로세싱 실체들에 의해 공유된 데이터 엘리먼트를 제거하라는 보류중인 요구를 갖는 상기 프로세싱 실체의 비트맵을 표현하는 분배된 인디케이터를, 상기 프로세싱 실체들 각각과 연관시키는 단계;

상기 프로세싱 실체들이 토큰 프로세싱에 참여하기 이전에 상기 프로세싱 실체들 각각에서 상기 분배된 인디케이터를 프로세싱하는 단계; 및

상기 분배된 인디케이터에 의해 허가된 경우에만 상기 프로세싱 실체에서 토큰 프로세싱을 수행하는 단계를 포함하고,

상기 분배된 인디케이터를 프로세싱하는 단계는, 상기 프로세싱 실체들 중 하나의 이웃 프로세싱 실체에서의 상기 분배된 인디케이터의 값에 따라 상기 분배된 인디케이터를 수정하는 단계와, (1) 현재 유예 기간과 연관된 제거 요구가 있는지의 여부에 따라, 상기 평가 프로세싱 실체에 대응하는 비트를 1로 설정하거나, (2) 현재 유예 기간과 연관된 어떠한 제거 요구도 없었는지의 여부에 따라, 상기 평가 프로세싱 실체에 대응하는 비트를 0으로 설정하는 방식으로, 상기 평가 프로세싱 실체에 대응하는 비트를 설정하는 단계를 포함하고,

이렇게하여, 상기 분배된 인디케이터가 불필요한 토큰 프로세싱을 허가하지 않는 경우 불필요한 토큰 프로세싱이 방지될 수 있는 것인, 유예 기간 탐지 방법.

명 세 서

기 술 분 야

<1> 본 발명은 데이터 리소스가 각각의 소비자에 관한 데이터 무결성 및 일관성을 보존한 채 데이터 동시사용 소비자들간에 공유되는 컴퓨터 시스템 및 방법에 관한 것이다. 더욱 상세하는, 본 발명은 로크-프리 데이터 판독 동작이 데이터 갱신 동작과 동시에 실행되는, "판독-복사 갱신(read-copy update)"으로 알려진 상호 배제 메커니즘에 대한 개선에 관련된다.

배 경 기 술

<2> 배경기술로서, 판독-복사 갱신은 공유된 데이터가 동시적으로 갱신(수정, 삭제, 삽입등)될 수 있도록 하는 한편, 로크, 메모리 장벽, 아토믹 명령어, 또는 다른 고비용 연산 동기화 메커니즘을 사용하지 않거나, 공유 메모리로의 기입을 행하지 않고, 판독을 위해 그 공유된 데이터에 액세스되게 하는 상호 배제 기술이다. 이 기술은 공유된 데이터 셋트를 액세스하는 판독 동작(판독기)의 수가 갱신 동작(갱신기)의 수에 비해 많은 멀티프로세서 컴퓨팅 환경에 양호하게 적용되고, 이 기술에서 각각의 판독 동작을 위한 기타 상호 배제 기술(로크와 같은)을 채용하는 부담 비용은 높아지게 된다. 예로서, 갱신 동작은 수분 마다 기껏 1번씩 행해지지만 탐색동작은 초당 수천번 행해지는 네트워크 라우팅 테이블은 판독면에서 로크 획득이 매우 부담이 많게 되는 경우이다.

<3> 판독-복사 갱신 기술은 두 단계(phase)로 데이터 갱신을 구현한다. 첫 번째(initial update; 초기 갱신) 단계에서, 실제 데이터 갱신은 갱신되고 있는 데이터의 두 가지 외형(views)을 일시적으로 보존하는 방식으로 수행된다. 그중 한 외형은 데이터를 현재 참조하는 동작을 위해 유지되는 오래된(pre-update; 갱신 이전) 데이터 상태이다. 다른 한 외형은 갱신에 뒤이어 데이터를 액세스하는 동작을 위해 이용될 수 있는 새로운(post-update; 갱신 이후) 데이터 상태이다. 두번째(deferred update; 연기된 갱신)단계에서, 오래된 데이터 상태는, 모든 실행 동작이 갱신 전 데이터에 대한 참조(references)를 더 이상 유지하지 않을 것이라는 것을 보장하기에 충분히 긴 "유예기간(grace period)"에 뒤이어 제거된다.

<4> 도 1a-1d는 판독-복사 갱신을 이용하여 데이터 엘리먼트(A,B 및 C)의 그룹 에서 데이터 엘리먼트(B)를 수정하는 것을 예시한다. 데이터 엘리먼트(A,B 및 C)는 비순환 방식으로 트래버스하는 단식 링크된 리스트(singly-linked list)로 배열되는 데, 각각의 데이터 엘리먼트는 데이터의 일부 아이템을 저장하는 것 이외에 상기 리스트에서 다음 엘리먼트에 대한 포인터(또는 최종 엘리먼트에 대한 널(NULL) 포인터)를 포함하고 있다. 광역 포인터(도시되지 않음)는 그 리스트의 첫번째 멤버인 데이터 엘리먼트(A)를 포인팅하는 것으로 가정된다. 당업자는 데이터 엘리먼트(A,B 및 C)가 C-언어의 "struct"변수에 의해 정의된 데이터 구조를 포함하는 임의의 다양한 종래의 프로그래밍 구조체를 사용하여 구현될 수도 있지만, 이에 한정되지 않음을 인식할 것이다.

<5> 도 1a-1d의 데이터 엘리먼트 리스트는 복수의 동시사용 판독기에 의해 트래버스되고(로킹되지 않고) 상기 리스트내의 데이터 엘리먼트를 삭제, 삽입 또는 수정하는 갱신기에 의해 때때로 갱신된다고 가정된다. 도 1a에서, 데이터 엘리먼트(B)는 그 아래에 수직 방향 화살표로 도시된 바와 같이, 판독기(r1)에 의해 참조되고 있다. 도 1b에서, 갱신기(u1)는 데이터 엘리먼트(B)를 수정함에 의해 링크된 리스트를 갱신하길 바라고 있다. r1이 데이터 엘리먼트(B)를 참조하고 있다는 사실(이는 r1이 기능하지 못하게 할 수 있다)에 관계없이 데이터 엘리먼트(B)를 단순히 갱신하는 대신에, u1은 데이터 엘리먼트(B)의 갱신된 버젼(도 1c에서 데이터 엘리먼트 B'로서 도시됨)을 발생시키고 발생된 데이터 엘리먼트를 링크된 리스트에 삽입하는 한편 데이터 엘리먼트(B)를 보존한다. 이것은 u1이 스핀로크를 획득하고, B'를 위한 새로운 메모리를 할당하고, B의 내용을 B'에 복사하고, 필요에 따

라 B'를 수정하고, 포인터가 B'를 포인팅하도록 포인터를 A로부터 B로 갱신하고, 스핀로크를 해제함으로써 행해진다. 링크된 리스트를 횡단하는, 판독기(r2)와 같은, 모든 후속(갱신 후) 판독기는 B'를 만나게됨으로써 갱신 동작의 효과를 알게 된다. 반면에, 오래된 판독기(r1)는 영향을 받지 않는데 이는 B의 갱신전 본래 버젼 및 C를 향한 포인터가 유지되기 때문이다. r1이 이제 쓸모없게 된 데이터를 판독하고 있을지라도, 이러한 판독은 데이터 엘리먼트가 컴퓨터 시스템 외부의 컴포넌트의 상태를 추적하는 경우와 같은, 허용될 수 있는 경우 및 통신 지연으로 인한 오래된 데이터를 반드시 허용해야만 하는 여러 경우가 있다.

<6> 갱신에 뒤이어지는 어떤 후속 타임에서, r1은 링크된 리스트의 트래버스를 계속하고 B에 대한 참조를 하지않고 다른 것을 향해 이동하게 될 것이다. 또한, 그밖의 어떤 판독기 프로세스도 B를 액세스할 권한을 갖지않는 타임이 있게된다. 상기 설명한 바와 같은 유예 기간 만료를 나타내는, 그 시점(point)에서, 도 1d에 도시된 바와 같이, u1이 B를 자유롭게 놓아준다.

<7> 도 2a-2c는 판독-복사 갱신을 사용하여 데이터 엘리먼트(A, B 및 C)로 이루어 진 단식 링크된 리스트에서 데이터 엘리먼트(B)를 삭제하는 것을 예시한다. 도 2a에 도시된 바와 같이, 판독기(r1)는 현재 B를 참조하고 있고 갱신기(u1)는 B를 삭제하길 바라고 있는 것으로 가정된다. 도 2b에 도시된 바와 같이, 갱신기(u1)는 포인터를 A로부터 B로 갱신하고 따라서 A는 이제 C를 포인팅한다. 이러한 방식에서, r1은 영향을 받지 않지만 후속하는 판독기(r2)는 삭제의 결과를 알게된다. 도 2c에 도시된 바와 같이, r1은 후속하여 B에 대한 참조를 이동시키게 되고, 따라서 B1이 유예 기간 만료에 뒤이어 자유롭게 되어질 수 있게 한다.

<8> 판독-복사 갱신 메커니즘에 대하여, 유예 기간은 판독-복사 갱신에 의해 보호된 데이터 엘리먼트에 대한 액세스 권한을 갖는 모든 실행중인 프로세스가 데이터 엘리먼트에 대한 참조를 유지하거나, 데이터 엘리먼트에 대한 로크를 표명하거나, 또는 데이터 엘리먼트 상태에 대해 어떠한 가정도 더 이상 행할 수 없는 "정적 상태(quiescent state)"를 지난 시점을 나타낸다. 종래에, 운영체제를 위한 커널 코드 경로, 컨텍스트(프로세스) 스위치, 유휴 루프, 및 사용자 모드 실행은 모두(본원에 리스트되지 않을 수 있는 기타 동작일 수 있는 바와 같은)임의의 주어진 CPU에 대한 정적 상태를 나타낸다.

<9> 도 3에서, 4개의 개별 CPU에서 실행하고 있는 4개 프로세스(0, 1, 2 및 3)는 정적 상태를 주기적으로 통과하는 것으로 도시되어 있다.(2중 수직 바아 표현됨)에). 유예 기간(점선 수직 라인으로 도시됨)은 4개 프로세스(0, 1, 2 및 3) 모두가 하나의 정적 상태를 통과한 타임 프레임을 포함한다. 4개 프로세스가 도 1a-1d 또는 도 2a-2c의 링크된 리스트를 트래버스한 판독기 프로세스들이면, 유예 기간 이전에 오래된 데이터 엘리먼트(B)에 대한 참조를 갖는 이들 프로세스들의 어느 것도 유예 기간에 뒤이어 참조를 유지할 수 없다. 이들 프로세스에 의해 수행된 모든 유예 기간 후 탐색은 갱신기에 의해 삽입된 링크를 뒤따름으로써 B를 바이패스 하게 된다.

<10> 발명의 명칭이 "실행 이력 및 스레드 모니터링을 이용하는 멀티프로세서 시템에서 감소된 오버헤드 상호-배제 및 유지보수 코히어런시를 달성하기 위한 방법 및 장치(Apparatus And Method For Achieving Reduced Overhead Mutual-Exclusion And Maintaining Coherency In A Multiprocessor System Utilizing Execution History And Thread Monitoring)"이고, 공동 양도된 미국 특허 제 5,727,209호에 설명된 바와 같은 콜백 프로세싱 기술의 사용을 포함하지만 이에 한정되지 않는, 연기된 데이터 갱신을 유예 기간에 뒤이어 구현하는 데에 사용될 수 있는 여러 다양한 방법이 있다. 미국 특허 제5,727,209호의 내용은 참조문헌으로서 본원 명세서에 통합되었다.

<11> 콜백 프로세싱 기술은 공유된 데이터 엘리먼트의 갱신기가 갱신되고 있는 데이터의 새로운 외형을 생성하는 초기(제1 단계) 데이터 갱신 동작을 수행하고, 그후 갱신되고 있는 데이터의 오래된 외형을 제거하는 연기된 (제2 단계) 데이터 갱신 동작을 수행하기 위한 콜백 기능을 지정하는 것을 의도한다. 갱신기는 판독-복사 갱신 서브시스템이 유예 기간의 끝(end)에서 실행될 수 있도록 콜백 기능(이하에서 "콜백"으로 언급됨)을 판독-복사 갱신 서브시스템에 등록할 것이다. 판독-복사 갱신 서브시스템은 각각의 프로세서의 현재 유예 기간이 만료된 때를 탐지하기 위해 각각의 프로세서에 대한 보류중인(pending) 콜백을 추적하고 프로세서 마다의 정적 상태 액티비티를 모니터한다.

<12> 판독-복사 갱신의 성공적인 구현은 유예 기간의 길이를 추정하기 위한 효과적인 메커니즘을 필요로 한다. 판독-복사 갱신 구현중 하나의 중요한 구현 부류는 유예 기간 토큰을 소유하고 있는 프로세에 대해 유예 기간의 끝에 도달하였다는 것을 표명하기 위해 유예 기간 토큰을 한 프로세서로부터 다른 프로세로 전달한다. 유예 기간 토큰은 프로세들간에 명시적으로 전달되는 식별값일 수 있다. 그러나, 이 기술을 사용하는 경우 두 개의 기입 액세스-유예 기간 토큰을 유예 기간 토큰의 현재 소유자로부터 제거하는 하나의 액세스 및 유예 기간 토큰을 유예 기간 토큰의 새로운 소유자에게 전달하는 또다른 액세스-를 필요로 한다. 유예 기간 토큰을 핸들링하는 더욱 효과적인 방식은 프로세서에 따른(per-processor) 정적 상태 카운터 및 연관된 폴링 메커니즘을 암묵적으로 전

달하는 것이다. 이 기술에 따르면, 프로세서가 정적 상태를 통과할 때 마다 그 폴링 메커니즘은 현재 프로세서의 최종 유예 기간 이후 이웃하는 프로세서의 정적 상태카운터가 변경되었는 지를 알기 위해 이웃 프로세서의 정적 상태 카운터를 검사한다. 정적 상태 카운터가 변경되었다면, 현재 프로세서는 토큰을 최종적으로 가졌었던 이후 새로운 유예 기간이 경과되었다고 결정한다. 현재 프로세서는 자신의 보류중인 콜백을 실행하고 그후 자신의 정적 상태 카운터를 이웃 프로세서의 값보다 높은 값으로 점진 증분식으로 변경한다. 다음 프로세서는 그후 현재 프로세서의 변경된 카운터 값을 보고, 자신의 보류중인 콜백을 프로세스하고, 그리고 자신 소유의 카운터를 증가시킨다. 이 시퀀스는, 궁극적으로 유예 기간 토큰이 라운드-로빈 방식으로 모든 프로세서를 통하여 행함으로써, 계속된다.

<13> 유예 기간 토큰이 구현되는 방식과 무관하게, 각각의 프로세서는 유예 기간 토큰을 수신하는 경우 단지 콜백만을 프로세스한다. 유예 기간 토큰은 현재 보유자인 프로세서에 도달하기 전에 모든 다른 프로세서를 통하여 진행해야 하는 한, 현재 프로세서는 자신이 유예 기간 토큰을 소유했던 최종 시간 이후 정적 상태를 통과했다는 것이 언제나 보장되고, 이에따라, 유예 기간이 경과되었음을 보증한다.

<14> 토큰 조작을 이용한 유예 기간 탐지는 프로세서가 자신의 정적 상태를 통과함에 따라 프로세서 사이클을 소비하기 때문에, 보류중인 콜백이 판독-복사 갱신 서브시스템에 있지 않다면 그러한 사이클을 소비하는 오버헤드를 초래하는 것은 바람직하지 않다. 이 러한 이유로, 효과적인 토큰-기반 판독-복사 갱신 구현은 판독-복사 갱신 서브시스템이 유휴상태에 있는 지를 결정하기 위해 유예 기간 토큰 프로세싱 전에 테스트되는 공유 인디케이터(즉, 광역 변수)를 사용한다. 판독-복사 갱신 서브시스템이 유휴상태에 있다면, 유예 기간 토큰은 전달될 필요가 없고 연관된 프로세싱 오버헤드는 방지될 수 있다. 공유 인디케이터는 전형적으로 보류중인 콜백의 수에 대한 카운트이다. 콜백이 주어진 프로세서에 등록될 때 마다, 공유 인디케이터는 새로운 콜백을 반영하도록 조작된다. 그후, 콜백이 프로세스되면, 공유 인디케이터는 판독-복사 갱신 서브시스템으로부터 콜백의 제거됨을 반영하도록 조작된다.

<15> 보류중인 콜백의 존재에 대해 테스트하기 위해 공유 인디케이터를 사용하는 단점은 아토믹 명령, 로크 또는 상대적으로 고비용이 드는 기타 상호 배제 메커니즘이 복수의 프로세서에 의해 공유 인디케이터로 동작을 동기화하기 위해 공유 인디케이터가 조작될 때 마다 호출되어야만 한다는 것이다. 더욱이, 각각의 프로세서에 의한 공유 인디케이터의 종래의 하드웨어 구현에 의한 캐싱은 통신 캐쉬 미스 또는 캐쉬 라인이 고장나는 결과로 되는 경향이 있다. 비트맵 인디케이터의 경우에, 다수의 프로세서가 무리없이 수용될 수 없다는 추가의 단점이 있다.

<16> 본 발명은 상기한 바와 같은 문제점을 해결하는 방안을 설명한다. 특히, 보류중인 콜백 상태에 대한 공유 인디케이터의 오버헤드를 초래하지 않고 불필요한 유예 기간 토큰 프로세싱을 방지하는 새로운 판독-복사 갱신 유예 기간 탐지 기술이 필요로 된다.

발명의 상세한 설명

<17> 판독-복사 갱신 서브시스템 또는 데이터 엘리먼트에 대한 기존 참조가 제거될 때 까지 공유된 데이터 엘리먼트의 제거를 연기시킬 것을 필요로 하는 기타 프로세싱 환경에서 아토믹 명령어없이 유예 기간을 탐지하는 방법, 시스템 및 컴퓨터 프로그램 제품에 의해, 상기한 바와 같은 종래기술의 문제점이 해결되고 종래기술에 비해 진보된다. 유예 기간 탐지 방법은 공유된 데이터 엘리먼트에 대한 액세스를 공유하는 프로세싱 실체들간에 순환되어야 할 토큰을 확립하는 단계를 포함한다. 유예 기간은 토큰이 프로세싱 실체들 사이를 왕복할때 마다 경과하는 것으로 결정될 수 있다. 공유된 데이터 엘리먼트 또는 프로세싱 실체에 의해 공유된 기타 데이터 엘리먼트에 대해 제거 프로세싱을 수행할 필요가 있는 지(예로서, 본 발명이 콜백-기반 판독-복사 갱신 시스템으로 구현되었다면 콜백 프로세싱을 보장하는 보류중인 콜백이 있는지)를 지시하는 분배된 인디케이터는 프로세싱 실체의 각각과 연관된다. 분배된 인디케이터는 프로세싱 실체에서 토큰 프로세싱에 참여하기 전에 각각의 프로세싱 실체에서 프로세스된다. 토큰 프로세싱은 분배된 인디케이터에 의해 허가(warrant)된 경우에만 프로세싱 실체에서 수행된다. 이러한 방식으로, 불필요한 토큰 프로세싱은 분배된 인디케이터가 그러한 불필요한 토큰 프로세싱을 허가하지 않는 경우에 방지될 수 있다.

<18> 발명의 대표적인 실시예에서, 분배된 인디케이터는 프로세싱 실체와 연관된 캐쉬 메모리에 로컬 변수로서 저장된다(그리고 종래의 캐쉬 코히어런스 메커니즘을 통해 프로세싱하는 도중에 한 캐쉬 메모리로부터 다른 캐쉬 메모리로 복제된다). 이러한 실시예에서, 분배된 인디케이터는 디자인 선호사항에 따라 상이한 정보 유형을 표현할 수 있다. 예를들어, 분배된 인디케이터는 프로세싱 실체에 의해 공유된 데이터 엘리먼트에 대한 갱신을 수행하라는 보류중인 요구, 전체 갱신 횟수, 또는 보류중인 갱신 요구를 갖는 프로세싱 실체를 식별하는 비트맵을

갖는 프로세싱 실체의 수를 선언적으로 표현할 수 있다.

<19> 다양한 프로세싱 실체에 의해 분배된 인디케이터에 대해 행해진 변경의 전달은 또한 디자인 선호사항에 따라 상이한 방식으로 수행될 수 있다. 대표적인 실시예에서, 프로세싱 실체는 주기적으로 이웃 프로세싱 실체에 의해 유지된 분배된 인디케이터를 참고하고, 현재 프로세싱 실체에서 데이터 엘리먼트 제거 요구 액티비티(예로서 콜백 등록)에서의 변경을 반영하기 위해 필요로 되는 바에 따라 분배된 인디케이터를 조정한다. 데이터 엘리먼트에서의 제거 요구 액티비티 변경이 있었다면 토큰의 순환을 보장하기 위해 프로세싱 실체중의 하나에서 보류중인 데이터 엘리먼트 제거 요구의 임계 횟수가 있는 지의 여부와 같은, 다양한 결정 요인을 포함할 수 있다. 대안으로, 그러한 결정은 프로세싱 실체중의 하나에 임의의 보류중인 데이터 엘리먼트 제거 요구가 있는 지의 여부에 기초한다.

실 시 예

<36> 대표적인 실시예에 대한 상세한 설명

<37> 이제, 마찬가지의 참조번호가 첨부도면에 포함된 모든 도의 형태에서 마찬가지의 구성요소를 나타내는, 본원의 첨부도면을 참조하면, 도 4는 본 발명이 구현될 수 있는 대표적인 컴퓨팅 환경을 예시한다. 상세히는, 복수 개의 프로세서(4_1, 4_2, . . . 4_n)가 공통 버스(6)에 의해 공유 메모리(8)에 연결되어 있는 대칭형 멀티프로세서(SMP;Symmetrical Multiprocessor) 컴퓨팅 시스템(2)이 도시되어 있다. 프로세서(4_1, 4_2, . . . 4_n)의 각각은 종래의 캐쉬 메모리(10_1, 10_2, . . . 10_n) 및 캐쉬 컨트롤러(12_1, 12_2, . . . 12_n)와 각각 연결되어 있다. 종래의 메모리 컨트롤러(14)는 공유 메모리(8)와 연관된다. 컴퓨팅 시스템(2)은 SMP 환경에서의 사용을 위해 응용된 단일 멀티태스킹 운영 체제의 관리하에 있는 것으로 가정된다.

<38> 또한 커널 또는 사용자 모드 프로세스, 스레드, 또는 기타 실행 컨텍스트내에서 실행된 갱신 동작은 공유 메모리(8)에 저장된 공유된 데이터 셋트(16)에 대해 갱신을 수행한다. 참조번호(18_1, 18_2, . . . 18_n)는 여러 프로세서(4_1, 4_2, . . . 4_n)에 의해 주기적으로 실행하는 개별 데이터 갱신 동작(갱신기)을 도시한다. 발명의 배경 단락에 의해 설명된 바와 같이, 데이터 갱신기(18_1, 18_2, . . . 18_n)에 의해 수행된 갱신은 링크된 리스트의 엘리먼트를 수정하는 것, 새로운 엘리먼트를 링크된 리스트에 삽입하는 것, 링크된 리스트로부터 삭제하는 것, 및 다수의 기타 유형의 동작을 포함할 수 있다. 이러한 갱신을 용이하게 하기 위해, 여러 프로세서(4_1, 4_2, . . . 4_n)는, 판독-복사 갱신 인스턴스(20_1, 20_2, . . . 20_n)의 운영 체제 기능의 일부로서 각각의 인스턴스(20_1, 20_2, . . .20_n)를 주기적으로 실행함에 의하는 바와 같이, 판독-복사 갱신(RCU; read-copy udate) 서브시스템(20)을 구현하도록 프로그램된다. 도면에 도시되지 않았을 지라도, 프로세서(4_1, 4_2, . . . 4_n)는 또한 공유된 데이터 셋트(16)에 대해 판독 동작을 실행한다. 이러한 판독 동작은 통상적으로, 판독-복사 갱신의 사용의 기저를 이루는 전제조건중의 하나인 이상, 갱신 동작보다 훨씬 빈번하게 수행되어질 것이다.

<39> 도 5에 도시된 바와 같이, 판독-복사 갱신 서브시스템 인스턴스(20_1, 20_2, . . . 20_n)의 각각은 콜백 등록 컴포넌트(22)를 포함한다. 콜백 등록 컴포넌트(22)는 갱신기(18_1, 18_2, . . . 18_n) 스스로에 의해 수행된 초기(제1 단계) 갱신에 뒤이은 연기된(제2 단계) 데이터 엘리먼트의 갱신을 위한 요구를 등록하기 위해 상기 갱신기에 의해 호출될 수 있는 판독-복사 갱신 서브시스템(20)에 대한 API(Application Program Interface;응용 프로그램 인터페이스)로서의 역할을 한다. 당업계에 공지된 바와 같이, 이들 연기된 갱신 요구는 오래된 쓸모없는 데이터 엘리먼트의 제거를 포함하고, 판독-복사 갱신 서브시스템(20)내에서 콜백으로서 취급된다. 판독-복사 갱신 서브시스템 인스턴스(20_1, 20_2, . . . 20_n)의 각각은 부가적으로 콜백 프로세싱 시스템(26)외에, 정적 상태 카운터 조작 및 폴링 메커니즘(24)(또는 토큰을 전달하기 위한 기타 기능)을 포함한다. 기능부(24,26)는 종래에서와 같이, 커널 스케줄러의 일부분으로서 구현될 수 있다.

<40> 프로세서(4_1, 4_2, . . . 4_n)와 각각 연관된 캐쉬 메모리(10_1, 10_2, . . . 10_n)는 정적 상태 카운터(28_1, 28_2, . . . 28_n) 및 하나 이상의 콜백 큐(30_1, 30_2, . . . 30_n)를 저장한다. 정적 상태 카운터(28_1, 28_2, . . . 28_n)는 프로세서(4_1, 4_2, . . . 4_n)들간에 유예 기간 토큰을 전달하기 위한 목적으로 정적 상태 카운터 조작 및 폴링 메커니즘(24)(토큰 조작기)에 의해 관리된다. 그밖의 어떤 다른 유형의 토큰 전달법이 사용된다면, 정적 상태 카운터(28_1, 28_2, . . . 28_n)는 필요로 되지 않게될 것이라는 것을 알 수 있다. 콜백 큐(30_1, 30_2, . . . 30_n)는 새로

운 콜백이 콜백 등록 컴포넌트(22)에 등록됨에 따라 새로운 콜백이 첨부(또는 프리펜드) 된다. 콜백 프로세싱 시스템(26)은 콜백 큐(30_1, 30_2, . . . 30_n)에서 참조된 콜백을 실행하고, 그후 콜백이 처리됨에 따라 콜백을 제거할 책임이 있다.

<41> 도 7 및 8은 대표적인 4 프로세서 시스템내의 프로세서가 정적 상태를 통과함에 따라 상기 프로세서들간에 유예기간 토큰을 전달하기 위해 정적 상태 카운터(28_1, 28_2, . . . 28_n)가 사용될 수 있는 방법을 나타낸다. 도 7의 각각의 컬럼은 주어진 시점에서 모든 프로세서 정적 상태 카운터에 대한 대표적인 값을 도시한다. 음영으로 나타낸 셀들은 대응 프로세서가 유예기간 토큰의 소유자임을 지시한다. 각각의 경우에, 유예기간 토큰의 소유자는 프로세서의 카운터가 최소값을 갖고 그 프로세서의 이웃이 토큰 소유자의 카운터 값에 대해 불연속을 나타내는 카운터 값을 갖는 프로세서이다.

<42> 도 7 및 8에 의해 나타내어 진 콜백 전달 기술은 당업계에 공지되어 있고 이 도면들은 따라서 "종래 기술"로서 표기되어 있다. 발명의 배경 단락에 의해 설명된 바와 같이, 주어진 프로세서는 이웃 프로세서중의 하나(예로서, 이웃 프로세서들의 수에 대해 모듈로(%) 연산한 값이, 프로세서 번호가 현재 프로세서 보다 1이 더 큰 이웃 프로세서)에 의해 유지된 정적 상태 카운터를 참조함에 의해 주어진 프로세서가 유예 기간 토큰을 소유하는 지를 알아보기 위해 검사한다. 이웃 프로세서의 정적 상태 카운터가 현재 프로세서의 최종 유예 기간 이후로 변경되지 않았다면(즉, 카운터 값에 어떠한 불연속도 없음), 현재 프로세서는 새로운 유예 기간이 아직 경과하지 않았다고 결정하고 정상적인 프로세싱을 재개한다. 이웃 프로세서의 정적 상태 카운터가 현재 프로세서의 최종 유예 기간 이후 변경되었으면(즉, 카운터 값에 불연속이 있음), 현재 프로세서는 새로운 유예 기간이 경과하였다고 결정한다. 현재 프로세서는 그 보류중인 콜백을 프로세스하고 자신의 고유한 정적 상태 카운터를 이웃 프로세서의 값 보다 1 더 크도록 증가시킴으로써, 스스로 카운터 값에서의 불연속이 되지 않도록 한다. 예로서, 도 7의 t=0에서, 최저 정적 상태 카운터 값(1)을 갖는 프로세서(3)는 프로세서(0)에서 불연속 카운터 값(4)을 만나게 된다. 이것은 프로세서(3)에게 프로세서(3)의 최종 유예 기간 이후로 프로세서(3)의 피어 프로세서에 의해 행해진 3개(4-1)의 정적 상태가 있어왔음을 의미한다. 프로세서(3)은 따라서 새로운 유예 기간이 경과되고 이제 유예 기간 토큰을 갖게되었다고 결론짓는다. 프로세서(3)은 콜백 프로세싱을 수행하고 자신의 정적 상태 카운터 값을 4+1=5로 설정한다. t=1에서, 2인 정적 상태 카운터 값을 갖는 프로세서(2)는, 이제 프로세서(0)에서 불연속 카운터 값(5)을 만나게 된다. 프로세서(2)는 유예 기간 토큰을 갖게되었음을 결정하고, 콜백 프로세싱을 수행하고 자신의 카운터 값을 5+1=6으로 설정한다. 이러한 시퀀스를 계속 수행함으로써, 프로세서(1)는 t=2에서 유예 기간 토큰을 획득하고 프로세서(0)은 t=3에서 유예 기간 토큰을 획득한다. t=4에서 유예 기간 토큰은 프로세서(3)에게 반환되고 상기한 바와 같은 패턴은 반복된다. 부가적으로 도 8에 도시된 바와 같이, 프로세서(3)는 t=0, 4 및 8에서, 유예 기간 토큰(T)을 획득하게 되고, 프로세서(2)는 t=1 및 5에서, 유예 기간 토큰을 획득하게 되고, 프로세서(3)는 t=2 및 6에서, 유예 기간 토큰을 획득하게 되고, 그리고 프로세서(0)는 t=3 및 7에서, 유예 기간 토큰을 획득하게 된다.

<43> 발명의 배경 단락에 의해 설명된 바와 같이, 판독-복사 갱신 기술의 종래기술의 구현은 프로세싱을 필요로 하는 판독-복사 갱신 서브시스템에 보류중인 콜백이 있는 지를 지시하는 공유 인디케이터로서의 역할을 하는 광역 변수를 조작함에 의해 불필요한 토큰 프로세싱을 방지하는 방법을 추구한다. 예를들어, P. McKenney등에 의해 "Read Copy Update", Ottawa Linux Symposium(2002)에서 개시된 바와 같이, "rcu-sched"로서 알려진 판독-복사 갱신에 대한 리눅스 구현은 판독-복사 갱신 서브시스템에있는 보류중인 콜백 횟수의 카운트를 표현하는 공유 변수 **"rcu_pending"**을 사용한다. 리눅스 아토믹 증가 프리미티브 **"atomic_inc"**는 새로운 콜백이 함수 콜**"atomic_inc(&rcu_pending)"**에 의해 등록되는 경우 **rcu_pending**을 증가시키기 위해 호출된다. 리눅스 아토믹 감소 프리미티브 **"atomic_dec"**는 상기 콜백이 함수 콜**"atomic_dec(&rcu_pending)"**에 의해 프로세스된 후 **rcu_pending**을 감소시키기 위해 호출된다. "rcu-sched"는 도 7 및 8에 도시된 바와 같은 카운터-기반, 유예 기간 토큰 전달 체계를 사용하는 판독-복사 갱신 구현의 한 예임을 알아야 한다.

<44> 공유 인디케이터의 콜백 발생예상도(pendency)를 증가 또는 감소시키기 위해 아토믹 동작(또는 기타 동시성 제어 메커니즘)의 사용과 연관된 단점을 방지하기 위해, 본 발명은 대안 연구방법을 제안한다. 도 6에 도시된 바와 같이, 분배된 콜백 인디케이터(32)는 각각의 프로세서(4_1, 4_2, . . . 4_n)의 캐쉬 메모리(10)에 유지될 수 있고 판독-복사 갱신 서브시스템(20)에서의 변경을 반영하기 위해 로컬 변수로서 조작될 수 있다. 각각의 분배된 콜백 인디케이터(32)는 판독-복사 갱신 서브시스템(20)의 상태에 대한 표현을 제공한다. 각각의 판독-복사 갱신 서브시스템 인스턴스(20_1, 20_2, . . . 20_n)내의 연관된 콜백 인디케이터 핸들링 메커니즘(34)(도 6에도 도시되어 있음)은 그후 유예 기간 토큰 프로세싱이 필요로 되는 지를 결정하기 위해 로컬 분배된 콜백 인디케이터

(32)를 참고한다. 로컬 분배된 콜백 인디케이터(32)는 판독-복사 갱신 서브시스템이 유휴 상태에 있다는 것을 나타낼 수 있고, 이 경우 유예 기간 토큰은 전달될 필요가 없다. 반면에, 로컬 분배된 콜백 인디케이터(32)는 판독-복사 갱신 서브시스템에 보류중인으로 되는 콜백이 있음을 나타낼 수 있고, 그리고 유예 기간 토큰 프로세싱은 현재 프로세서에서 필요로 된다.

<45> 조건들이 판독-복사 갱신 서브시스템(20)내에서 변경됨에 따라 분배된 콜백 인디케이터(32)를 현재 상태로 유지하기 위해, 도 7 및 8의 유예 기간 토큰 전달 체계와 약간 유사한 전파 기술이 사용될 수 있다. 그 밖의 구현도 가능할 수 있다. 이전파 기술에 따라, 각각의 프로세서(4_1, 4_2, . . .4_n)가 정적 상태를 겪음에 따라, 그 프로세서의 콜백 인디케이터 핸들링 메커니즘(34)은, 현재 프로세서의 최종 유예 기간 토큰 이후로 로컬 콜백 이력에 대한 고려사항과 함께, 이웃하는 프로세서의 분배된 콜백 인디케이터(32)를 참고하고, 이웃 프로세서의 값에 따라 자신의 고유 로컬 콜백 인디케이터를 조정한다.

<46> 본 발명의 제1 실시예에서, 분배된 콜백 인디케이터(32)는 보류중인 콜백을 갖는 프로세서의 수에 대한 프로세서에 따른 카운터로서 구현된다. 이들 프로세서는 "콜백 프로세서"로서 칭해질 수 있고, 분배된 콜백 인디케이터(32)는 콜백 프로세서 카운터로서 여겨질 수 있다. 이 카운터를 조작하기 위해, 프로세서는 그 프로세서의 최종 유예 기간 이후로 자신의 로컬 콜백 상태에서 임의의 어떠한 변경이 있었는 지를 알어보기 위해 검사한다. 어떠한 변경도 발생되지 않았으면, 현재 프로세서의 카운터는 이웃 프로세서의 카운터의 값과 동일 값으로 설정되어 질 것이다. 프로세서의 콜백 이력이, 어떠한 로컬 콜백도 그 프로세서를 떠난 최종 유예 기간 토큰이 등록되지 않았었지만, 최종 유예 기간 이후로 필수적인 새로운 로컬 콜백 수가 등록되었음을 보여준다면, 현재 프로세서의 카운터는 이웃 프로세서의 카운터의 값 보다 1 크도록 증가되어질 것이다. 프로세서의 콜백 이력이 로컬 콜백이 그 프로세서를 떠난 최종 유예 기간 토큰이 등록되었었지만, 최종 유예 기간 이후로 필수적인 새로운 로컬 콜백 수가 등록되지 않았음을 보여준다면, 현재 프로세서의 카운터는 이웃 프로세서의 카운터의 값 보다 1 낮도록 감소되어질 것이다.

<47> 본 발명의 제2 실시예에서, 분배된 콜백 인디케이터(32)는 보류중인 콜백의 전체 수에 대한 지시를 추적하도록 구현된다. 이 경우, 분배된 콜백 인디케이터(32)는 콜백 카운터로서 여겨질 수 있다. 이 카운터를 조작하기 위해, 프로세서는 이 프로세서의 최종 유예 기간 이후로 등록되어 있는 로컬 콜백의 수와 프로세서를 떠난 최종 유예 기간 토큰이 등록되었었던 로컬 콜백의 수를 비교한다. 현재 프로세서의 카운터는 로컬 콜백의 순 이득 또는 손실을 반영하기 위해 일정한 조정으로 이웃 프로세서의 카운터의 값으로 설정된다.

<48> 본 발명의 제3 실시예에서, 분배된 콜백 인디케이터(32)는 보류중인 콜백을 갖는 프로세서들을 식별하는 비트맵으로서 구현된다. 비트맵을 조작하기 위해, 프로세서는 최종 유예 기간 토큰이 그 프로세서를 떠난 이후로 등록되어진 필수 갯수의 로컬 콜백이 있는 지를 결정한다. 필수 갯수의 로컬 콜백이 있다면, 현재 프로세서의 비트맵은 이웃 프로세서의 비트맵에 대응하도록 설정되지만, 현재 프로세서의 비트는 1로 설정된다. 다른방법으로는, 최종 유예 기간 이후로 필수 갯수의 로컬 콜백이 등록되지 않았으면, 비트맵내의 현재 프로세서의 비트는 0으로 설정된다. 이와 같은 구현의 한 단점은 큰 비트맵을 대응하여 프로세스를 처리할 필요로 인해 다수의 프로세서를 무리없이 유연하게 핸들링하지 않는다는 것이다.

<49> 도 9는 분배된 콜백 인디케이터(32)가 보류중인 콜백을 갖는 프로세서(4_1, 4_2, . . .4_n)수에 대한 카운트인 상기 설명한 제1 실시예에 따라 수행될 수 있는 프로세싱 단계의 대표적인 시퀀스를 예시한다. 도 9의 프로세스는 **"cbcpus"**("callback cpus"의 축약형)라 칭하는 프로세서에 따른 로컬 변수를 분배된 콜백 인디케이터로서 사용한다. 이 로컬 변수는 프로세싱을 필요로 하는 콜백을 갖는 프로세서에 대한 카운트이다. **"lastcbs"**("last callback"의 축약형)라 칭하는 또다른 프로세서에 따른 로컬 변수는 현재 프로세서는 최종 유예 기간 토큰이 그 현재 프로세서를 제외한 등록된 콜백을 갖는 지를 지시하는 플래그이다. **"numcbs"**("number of callbacks"의 축약형)라 칭하는 프로세서에 따른 제3의 변수는 최종 유예 기간 이후로 현재 프로세서에서 등록된 콜백의 수에 대한 카운트이다.

<50> 도 9의 단계(40)에서, n번째 프로세서의 콜백 인디케이터 핸들링 메커니즘(34)은 프로세서(n+1)의 **cbcpus**의 값을 획득한다(프로세서(n-1)는 소망하는 전파 방향에 좌우되어 사용될 수도 있다). 단계(42)에서, 프로세서의 (n)은 유예 기간을 시작하기 위한 기준을 충족시키는 임의의 새로운 콜백(**numcbs**)이 있는 지를 결정한다. 몇몇 경우에, 단일 콜백의 존재는 상기 기준을 충족시키게 된다. 그 밖의 경우엔, 유예 기간을 개시시키는 데에 필요한 콜백의 수를 지정하는 콜백 임계치와, 콜백 임계치에 도달되지 않은 경우에도 콜백 프로세싱을 트리거시키는 경과 시간 임계치를 확립함으로써, 콜백들을 배치 프로세스로 작업하는 것이 바람직하다. 단계(42)에서, 프로세싱을 필요로 하는 새로운 콜백이 있다면, 단계(44)에서, 현재 프로세서의 **cbcpus** 값은 이웃 프로세서의 **cbcpus**

값 보다 1이 더 큰 값에서 현재 프로세서의 **lastcbs** 값을 뺀 값으로 설정된다. **lastcbs** 값은 단계(46)에서 현재 프로세서상에서의 콜백이 유예 기간을 시작하기 위한 기준을 충족하는 경우에만 1로 설정된다. 단계(42)에서, 프로세싱을 필요로 하는 어떠한 새로운 콜백도 없다면, 단계(48)에서, 현재 프로세서의 **cbcpus** 값은 이웃 프로세서의 **cbcpus** 값과 같은 값에서 현재 프로세서의 **lastcbs** 값을 뺀 값으로 설정된다. **lastcbs** 값은 현재 프로세서상에 유예 기간을 시작하기 위한 기준을 충족하는 어떠한 새로운 콜백도 없는 경우에만 단계(50)에서 0으로 설정된다.

<51> 각각의 프로세서가 정적 상태를 통과하는 한편 상기한 프로세싱을 수행함에 따라, 새로운 콜백의 등록 또는 오래된 콜백의 프로세싱은 분배된 콜백 인디케이터(본 예에서 **cbcpus**)의 각각에 의해 고속으로 반영되어지게 된다. 전파된 분배된 콜백 인디케이터를 각각의 프로세서에서 테스팅함으로써, 잠재적으로 고비용이 드는 토큰 프로세싱은 유예 기간 토큰 순환을 보장하는 충분한 콜백들이 없는 경우에 방지될 수 있다. 도 10의 테이블은 대표적인 4-프로세서 시스템에서 상기한 바와 같은 프로세싱을 예시한다. 도 10은 도 7을 기반으로 하지만, 각각의 프로세서(0, 1, 2 및 3)에 대해, 각각의 테이블 엘리먼트의 좌측에 있는 유예 기간 토큰 및 각각의 테이블 엘리먼트의 우측에 있는 분배된 콜백 인디케이터(본 예에서 **cbcpus**)를 도시하고 있다. 음영으로 나타내어 진 셀들은 대응 프로세서가 유예 기간 토큰의 소유자라는 것을 지시한다. 각각의 경우에, 유예 기간 토큰의 소유자는 그 정적 상태 카운터가 최소값을 갖고 그 이웃 프로세서가 토큰 소유자의 카운터 값에 대해 불연속을 나타내는 카운터 값을 갖는 프로세서이다.

<52> 도 10에서 프로세서(3)는 프로세서(0)으로부터 유예 기간 토큰을 수신한다. 그러나, 프로세서(3)의 분배된 콜백 인디케이터가 0인 값을 갖기 때문에 어떠한 토큰 프로세싱도 일어나지 않는다. 분배된 콜백 인디케이터(32)는 콜백 프로세서(**cbcpus**)의 카운트인 본 예에서, 0 값은 프로세싱을 보장하는 필수 갯수의 콜백을 갖는 어떠한 프로세서도 없음을 의미한다. 프로세서(3)는 따라서 이 프로세서 그룹을 위한 판독-복사 갱신 서브시스템이 유휴상태에 있다고 결정한다. 도 10의 시간 t=1에서, 프로세서(2)는 프로세서(2)가 새로운 콜백 액티비티를 가지고 있다고 결정하고 그 분배된 콜백 인디케이터를 값 1로 설정한다. 프로세서(3)는 영향을 받지 않고(프로세서(3)은 본 예에 따라 콜백 인디케이터 액티비티를 위해 단지 프로세서(0)만을 감시하고 있으므로) 한편으로 어떠한 유예 기간 토큰 프로세싱도 수행하지 않는다. 시간 t=2에서, 프로세서(2)의 분배된 콜백 인디케이터 값은 프로세서(1)에 전파된다. 프로세서(3)는 영향을 받지 않고 한편으로 어떠한 유예 기간 토큰 프로세싱도 수행하지 않는다. 시간 t=3에서, 프로세서(1)의 분배된 콜백 인디케이터 값은 프로세서(0)에 전파된다. 프로세서(3)는 영향을 받지 않고 한편으로 어떠한 유예 기간 토큰 프로세싱도 수행하지 않는다. 시간 t=4에서, 프로세서(0)의 분배된 콜백 인디케이터 값은 프로세서(3)에 전파되어짐으로써, 프로세서(3)가 유예 기간 토큰 프로세싱을 수행하고 유예 기간 토큰을 프로세서(2)에 전달하게 한다. 시간 t=5에서, 프로세서(2)는 유예 기간 토큰 프로세싱을 수행하였고 유예 기간 토큰을 프로세서(1)에 전달하였다. 시간 t=6에서, 프로세서(1)는 유예 기간 토큰 프로세싱을 수행하였고 유예 기간 토큰을 프로세서(0)에 전달하였다. 또한, 프로세서(2)는 그 콜백들이 프로세스되고 그 분배된 콜백 인디케이터가 0으로 설정되었다고 결정한 것으로 가정된다. 시간 t=7에서, 프로세서(0)는 유예 기간 토큰 프로세싱을 수행하였고 유예 기간 토큰을 프로세서(3)에 전달하였다. 또한, 프로세서(2)의 분배된 콜백 인디케이터는 프로세서(1)에 전파되었다. 시간 t=8에서, 프로세서(3)는 유예 기간 토큰 프로세싱을 수행하였고 유예 기간 토큰을 프로세서(2)에 전달하였다. 또한, 프로세서(1)의 분배된 콜백 인디케이터는 프로세서(0)에 전파되었다. 도 9의 시스템에서 어떠한 새로운 콜백도 등록되지 않았다고 가정하면, 유예 기간 토큰은 프로세서(2)의 분배된 콜백 인디케이터가 0 이므로 이제 프로세서(2)에서 유휴상태에 있게될 것이다.

<53> 도 11은 상기 설명한 프로세싱을 요약하여 나타낸다. 이 요약은 프로세서(3)는 시간 t=0, 7에서 토큰(T)을 획득하게 됨을 보여준다. 이 토큰은 시간 t=1, 2 및 3 동안 프로세서(3)에서 유휴상태에 있게될 것이다. 프로세서(2)는 시간 t=4, 8에서 토큰을 획득하게 될 것이다. 프로세서(1)는 시간 t=5에서 토큰을 획득하게 될 것이다. 프로세서(0)는 시간 t=6에서 토큰을 획득하게 될 것이다.

<54> 이제 도 12를 참조하면, 도 9의 분배된 콜백 인디케이터 프로세싱에 대한 대안 예가 도시되어 있다. 이 대안예의 방법에 따라, 단계(42a)(도 9의 단계(42)에 대응함)에서는 임계치에 도달했는 지의 여부에 무관하게, **numcbs**가 논제로인 지를 질의한다. 단계(46a)(도 9의 단계(46)에 대응함)에서는 **numcbs**가 0 보다 큰 경우에만 **lastcbs**를 1로 설정한다. 단계(50a)(도 9의 단계(50)에 대응함)에서는 **numcbs**가 0인 경우에만 **lastcbs**를 0으로 설정한다. 이러한 대안 방법의 장점은 프로세서가 그 콜백 프로세싱 필요를 다른 프로세서의 유예 기간 토큰 순환에 "피기백"시키고 토큰을 계속 이동시키기 위해 단지 적은 수의 콜백을 구비할 수 있도록 한다는 것이다. 단점은 추가적인 유예 기간 탐지 동작들이 수행되는 결과로 될 수 있다는 것이다.

<55> 도 13은 분배된 콜백 인디케이터(32)가 보류중인 콜백의 수에 대한 카운트인 상기한 제2 실시예에 따라 수행될

수 있는 프로세싱 단계로 이루어 진 대표적인 시퀀스를 예시한다. 도 13의 프로세스는 분배된 콜백 인디케이터로서 **"cbspen"**("callback pending"의 축약형)라 칭하는 프로세서에 따른 로컬 변수를 사용한다. **"lastcbs"**("last callback"의 축약형)라 칭하는 프로세서에 따른 또다른 로컬 변수는 현재 프로세서는 현재 프로세서를 제외한 최종 유예 기간 토큰이 등록되어 있었던 콜백의 수를 지시하는 값이다. **"numcbs"**("number of callbacks"의 축약형)라 칭하는 프로세서에 따른 세번째 변수는 최종 유예 기간 이후로 현재 프로세서에서 등록된 콜백의 수에 대한 카운트이다.

<56> 도 13의 단계(60)에서, n번째 프로세서의 콜백 인디케이터 핸들링 메커니즘(34)은 프로세서(n+1)의 **cbspen**의 값을 획득한다(프로세서(n-1)은 소망하는 전파 방향에 좌우되어 사용될 수도 있다). 단계(62)에서, 현재 프로세서의 **cbspen** 값은 이웃 프로세서의 **cbspen** 값에, 현재 프로세서의 **numcbs** 값을 더한 값에서, 현재 프로세서의 **lastcbs** 값을 뺀 값으로 설정된다. 이 **lastcbs** 값은 단계(66)에서, **numcbs**로 설정된다.

<57> 도 14는 분배된 콜백 인디케이터(32)가 어느 프로세서가 보류중인 콜백을 갖는 지를 도시하는 비트맵인 상기한 제3 실시예에 따라 수행될 수 있는 프로세싱 단계들로 이루어 진 대표적인 시퀀스를 예시한다. 도 14의 프로세스는 분배된 콜백 인디케이터로서 **"cbcpumap"**("callback cpu map"의 축약형)라 칭하는 프로세서에 따른 로컬 비트맵 변수를 사용한다. **"numcbs"**("number of callbacks"의 축약형)라 칭하는, 또다른 프로세서에 따른 로컬 변수는 최종 유예 기간 이후로 현재 프로세서에서 등록된 콜백의 수에 대한 카운트이다.

<58> 도 14의 단계(80)에서, n번째 프로세서의 콜백 인디케이터 핸들링 메커니즘(34)은 프로세서(n+1)의 **cbcpumap**의 값을 획득한다(프로세서(n-1)는 소망하는 전파 방향에 좌우되어 사용될 수도 있다). 단계(82)에서, 프로세서(n)는 몇몇 확립된 임계치를 충족하는 임의의 새로운 콜백(**numcbs**)이 프로세서(n)에 등록되었는 지를 결정한다(예로서, 도 9에 대해 상기 설명한 바와 같이). 단계(82)에서 프로세싱을 필요로 하는 새로운 콜백이 있으면, 단계(84)에서 현재 프로세서의 **cbcpumap**는 프로세서(n+1)의 **cbcpumap**과 같도록 설정되지만, **cbcpumap**의 n번째 비트는 1로 설정된다. 단계(82)에서 프로세싱을 필요로 하는 어떠한 새로운 콜백도 없다면, 단계(86)에서 현재 프로세서의 **cbcpumap**의 값은 프로세서(n+1)의 그 값과 같도록 설정되지만, **cbcpumap**의 n번째 비트는 0으로 설정된다.

<59> 따라서, 아토믹 명령어를 필요로 하지 않고 다수의 프로세서를 무리없이 유연하게 처리하도록 구현될 수 있는 판독-복사 갱신 유예 기간 토큰 탐지를 위한 기술이 개시되었다. 상기 설명한 개념들은 데이터 프로세싱 시스템, 머신 구현 방법, 및 필요로 되는 기능을 수행하기 위해 데이터 프로세싱 시스템을 제어하는 데에 사용하기 위한 하나 이상의 데이터 저장 매체에 프로그래밍 수단이 기록되는 컴퓨터 프로그램 제품중 어느 것에서나 다양하게 구현될 수 있다. 그러한 프로그래밍 수단을 저장하기 위한 대표적인 데이터 저장 매체는 도 15에 참조번호 100으로 나타나 있다. 저장 매체(100)은 상용 소프트웨어 판매를 위해 통상적으로 사용되는 휴대형 광 저장 디스크 유형으로서 도시되어 있다. 이러한 저장 매체는 단독으로 또는 운영체제와 연관하여 또는 판독-복사 갱신 기능을 내장하고 있는 기타 소프트웨어 제품과 연관하여 본 발명의 프로그래밍 수단을 저장할 수 있다. 이 프로그래밍 수단은 또한 휴대형 자기 기록매체(플로피 디스크, 플래쉬 메모리 스틱등과 같은) 또는 컴퓨터 플랫폼에 통합된 드라이브 시스템(예를들어, 디스크 드라이브)과 결합된 자기 기록매체에도 저장될 수 있다.

<60> 본 발명의 다양한 실시예가 설명된 반면에, 다양한 변형 및 대안 실시예가 본 발명에 따라 구현될 수 있음이 명백하게 될 것이다. 따라서, 본 발명은 첨부된 특허청구범위 및 이와 등가인 기술사상에 따르는 것을 제외하곤 임의의 어떠한 방식으로도 제한되지 않아야 함을 알아야 한다.

도면의 간단한 설명

<20> 본 발명의 상기한 설명 내용, 기타 특징 및 이점들은, 첨부한 도면에 나타낸 바와 같이, 본 발명에 대한 이하의 실시예에 대한 상세한 설명으로부터 명백히 알 수 있을 것이다.

<21> 도 1a-1d는 종래의 판독-복사 갱신 메커니즘에 따라 데이터 엘리먼트 교체가 행해지는 데이터 엘리먼트의 링크된 리스트를 도해식으로 나타낸 도.

<22> 도 2a-2c는 종래의 판독-복사 갱신 메커니즘에 따라 데이터 엘리먼트 삭제가 행해지는 데이터 엘리먼트의 링크된 리스트를 도해식으로 나타낸 도.

<23> 도 3은 4개 프로세스가 정적 상태를 통과한 유예 기간을 예시하는 흐름도.

<24> 도 4는 본 발명이 구현될 수 있는 하나의 대표적인 환경을 표현하는 멀티프로세서 컴퓨터 시스템을 도시하는 기능 블록도.

<25> 도 5는 도 4의 멀티프로세서 컴퓨터 시스템내의 각각의 프로세서에 의해 구현된 판독-복사 갱신 서브시스템을 도시하는 기능 블록도.

<26> 도 6은 도 4의 멀티프로세서 컴퓨터 시스템내의 각각의 프로세서와 연관된 캐쉬 메모리를 도시하는 기능 블록도.

<27> 도 7은 판독-복사 갱신을 구현하는 가상 4-프로세서 데이터 처리 시스템에서의 대표적인 정적 상태 카운터 값을 도시하는 테이블을 나타낸 도.

<28> 도 8은 판독-복사 갱신 처리 동안 도 7의 4개 프로세서가 일정 시간대별로 유예 기간 토큰을 통과하는 바와 같은 도 7의 4개 프로세서를 도시하는 기능 블록도.

<29> 도 9는 보류중인 콜백을 갖는 프로세서의 카운트로서 구현된 분배된 콜백 인디케이터의 조작을 도시하는 흐름도.

<30> 도 10은 판독-복사 갱신을 구현하는 하이포세티컬 4-프로세서 데이터 처리 시스템내의 대표적인 정적 상태 카운터 값 및 분배된 콜백 인디케이터 값을 도시하는 테이블을 나타낸 도.

<31> 도 11은 판독-복사 갱신 프로세싱 동안 도 10의 4개 프로세서가 시간으로부터 시간으로 유예 기간 토큰을 통과하는 바와 같은 도 10의 4개 프로세서를 도시하는 기능 블록도.

<32> 도 12는 도 9의 흐름도의 흐름순서의 수정을 나타낸 흐름도.

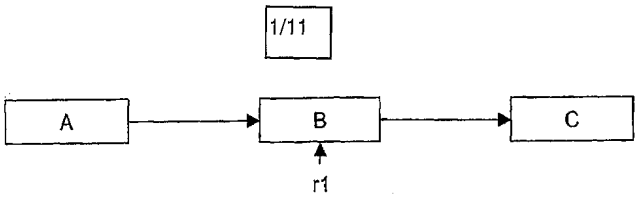
<33> 도 13은 보류중인 콜백의 카운트로서 구현된 분배된 콜백 인디케이터의 조작을 도시하는 흐름도.

<34> 도 14는 보류중인 콜백을 갖는 비트맵 식별 프로세서의 카운트로서 구현된 분배 콜백 인디케이터의 조작을 도시하는 흐름도.

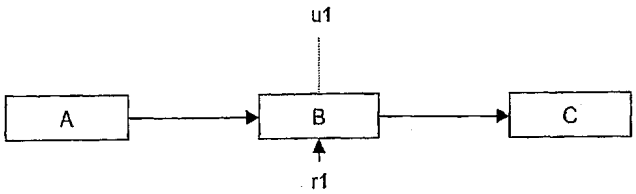
<35> 도 15는 본 발명에 따라 판독-복사 갱신 유예 기간 검출(탐지) 기능을 구현하기 위한 컴퓨터 프로그램 제품을 저장하는 데에 사용될 수 있는 저장 매체를 도해식으로 나타낸 도.

도면

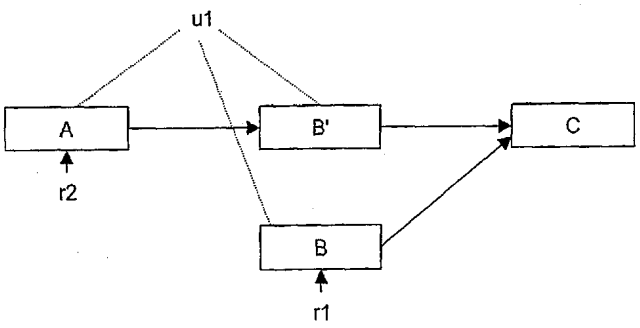
도면1a



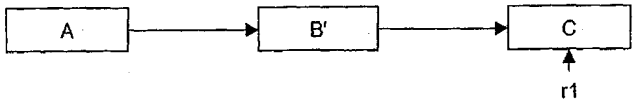
도면1b



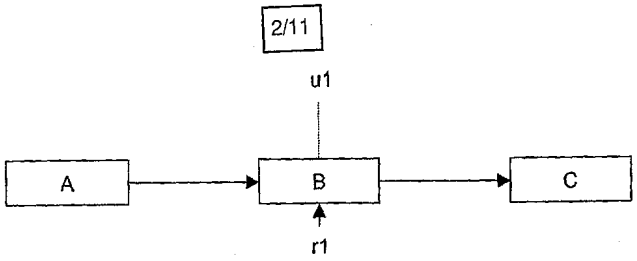
도면1c



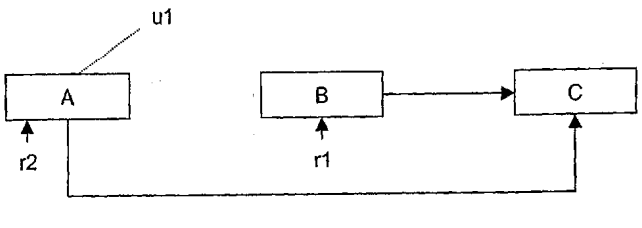
도면1d



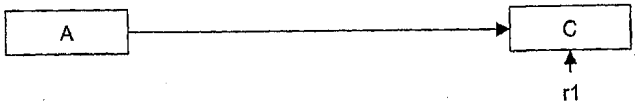
도면2a



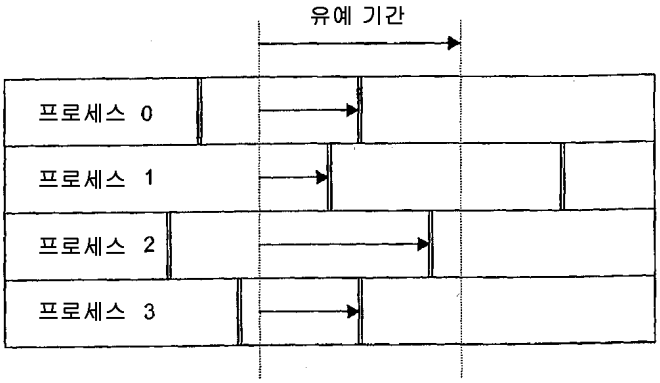
도면2b



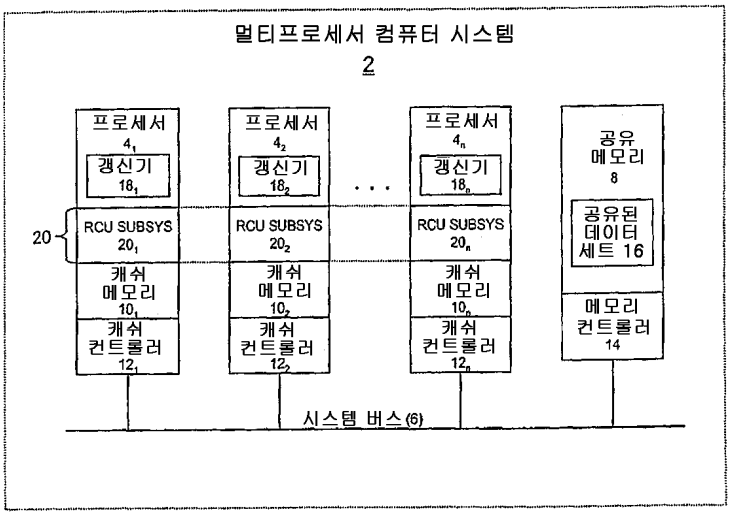
도면2c


A
C
r1

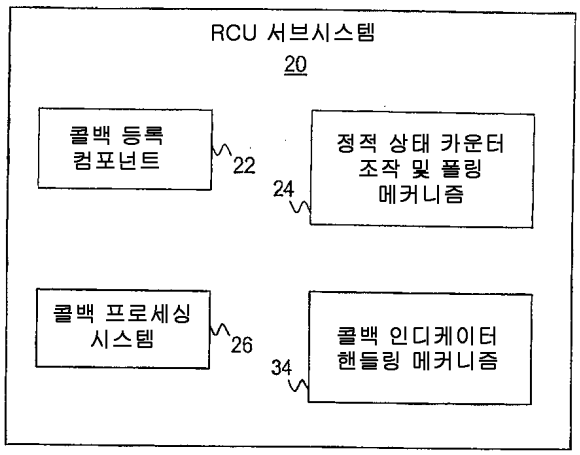
도면3


유예 기간
프로세스 0
프로세스 1
프로세스 2
프로세스 3

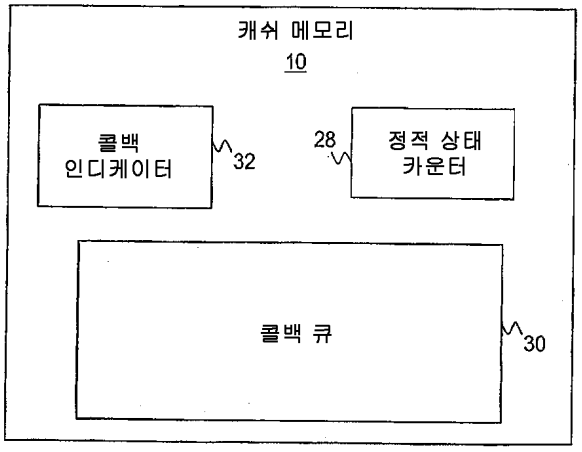
도면4


멀티프로세서 컴퓨터 시스템
2
프로세서 4₁
갱신기 18₁
RCU SUBSYS 20₁
캐쉬 메모리 10₁
캐쉬 컨트롤러 12₁
프로세서 4₂
갱신기 18₂
RCU SUBSYS 20₂
캐쉬 메모리 10₂
캐쉬 컨트롤러 12₂
프로세서 4ₙ
갱신기 18ₙ
RCU SUBSYS 20ₙ
캐쉬 메모리 10ₙ
캐쉬 컨트롤러 12ₙ
20
공유 메모리 8
공유된 데이터 세트 16
메모리 컨트롤러 14
시스템 버스 (6)

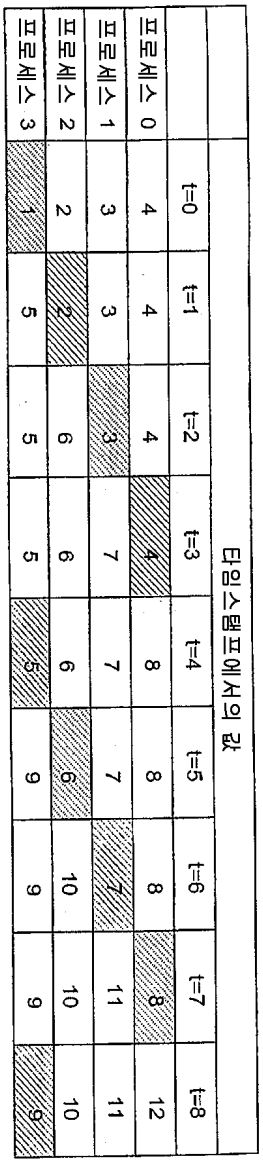
도면5


RCU 서브시스템
20
콜백 등록
컴포넌트
22
정적 상태 카운터
조작 및 폴링
메커니즘
24
콜백 프로세싱
시스템
26
콜백 인디케이터
핸들링 메커니즘
34

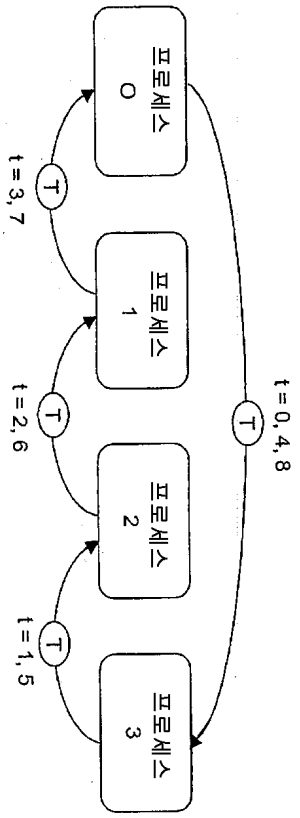
도면6


캐쉬 메모리
10
콜백
인디케이터
32
28
정적 상태
카운터
콜백 큐
30

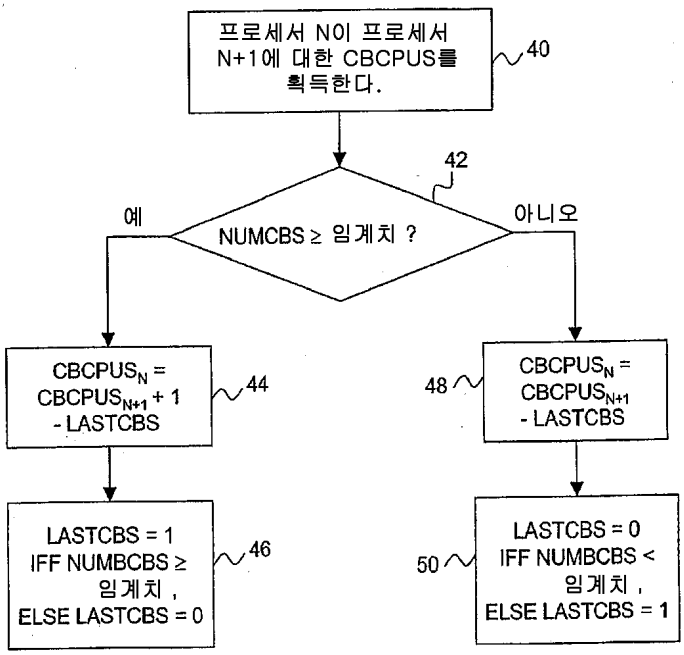
도면7


타임스탬프에서의 값
t=0
t=1
t=2
t=3
t=4
t=5
t=6
t=7
t=8
프로세스 0
4
4
4
4
8
8
8
8
12
프로세스 1
3
3
3
7
7
7
7
11
11
프로세스 2
2
2
6
6
6
6
10
10
10
프로세스 3
1
5
5
5
5
9
9
9
9

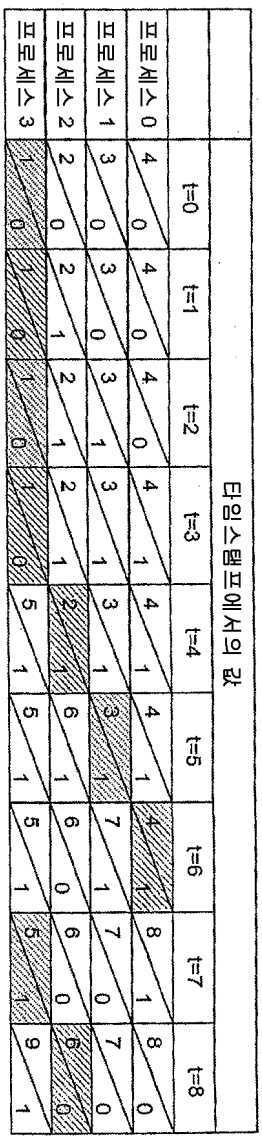
도면8


프로세스 0
프로세스 1
프로세스 2
프로세스 3
t = 3, 7
t = 2, 6
t = 1, 5
t = 0, 4, 8
T

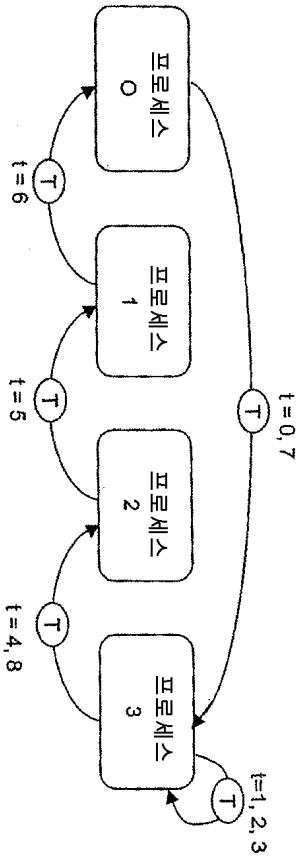
도면9


프로세서 N이 프로세서 N+1에 대한 CBCPUS를 획득한다.
40
42
NUMCBS ≥ 임계치 ?
예
아니오
CBCPUS_N = CBCPUS_N+1 + 1 - LASTCBS
44
LASTCBS = 1 IFF NUMBCBS ≥ 임계치, ELSE LASTCBS = 0
46
48
CBCPUS_N = CBCPUS_N+1 - LASTCBS
LASTCBS = 0 IFF NUMBCBS < 임계치, ELSE LASTCBS = 1
50

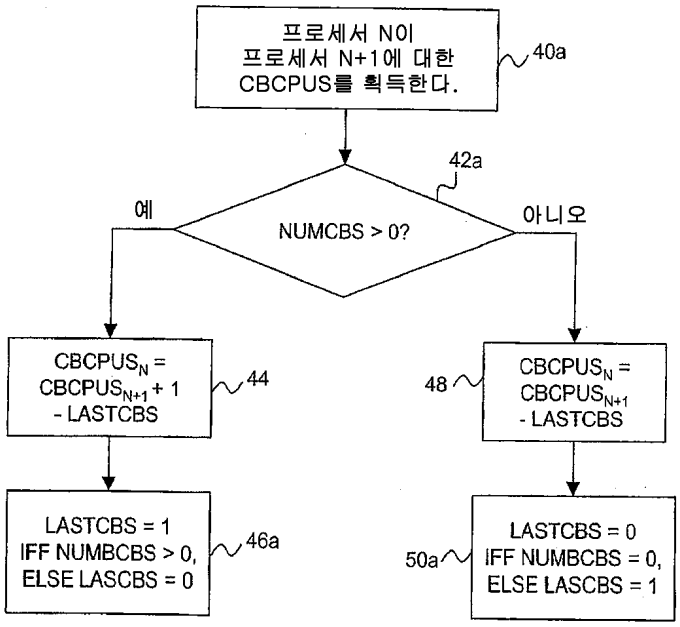
도면10



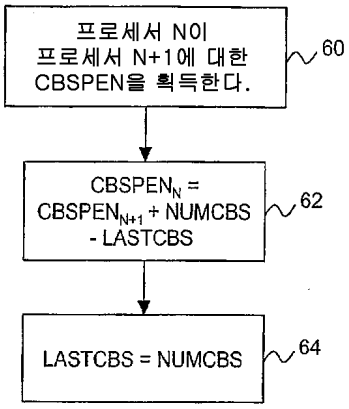
도면11


프로세스 0
프로세스 1
프로세스 2
프로세스 3
t = 6
t = 5
t = 4, 8
t = 0, 7
t=1, 2, 3
T

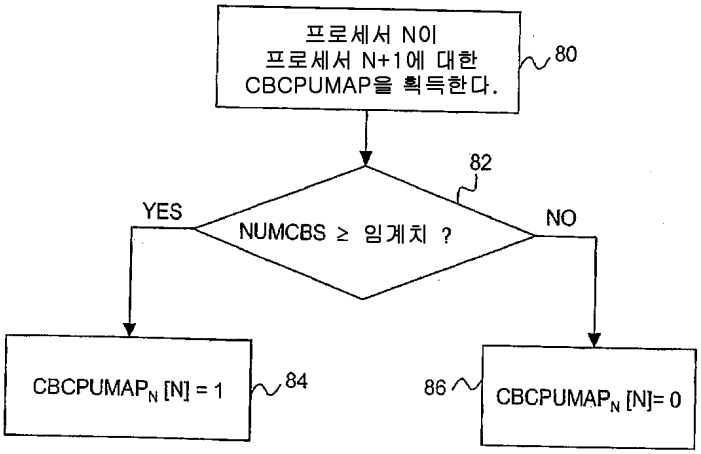
도면12


프로세서 N이 프로세서 N+1에 대한 CBCPUS를 획득한다.
40a
42a
NUMCBS > 0?
예
아니오
CBCPUS$_N$ = CBCPUS$_{N+1}$ + 1 - LASTCBS
44
LASTCBS = 1 IFF NUMBCBS > 0, ELSE LASCBS = 0
46a
48
CBCPUS$_N$ = CBCPUS$_{N+1}$ - LASTCBS
LASTCBS = 0 IFF NUMBCBS = 0, ELSE LASCBS = 1
50a

도면13



도면14



도면15