(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2009-0005921
(43) 공개일자 2009년01월14일

(51) Int. Cl.
G06F 9/46 (2006.01) *G06F 9/50* (2006.01)
(21) 출원번호 10-2007-0069353
(22) 출원일자 2007년07월10일
심사청구일자 2007년07월10일

(71) 출원인
삼성전자주식회사
경기도 수원시 영통구 매탄동 416
(72) 발명자
최규상
서울 서초구 서초동 1671-5 서초리시온 620호
임채석
경기 수원시 영통구 영통동 벽적골8단지아파트 848-705
이시화
서울 강남구 일원본동 샘터마을아파트 109동 304호
(74) 대리인
리앤목특허법인

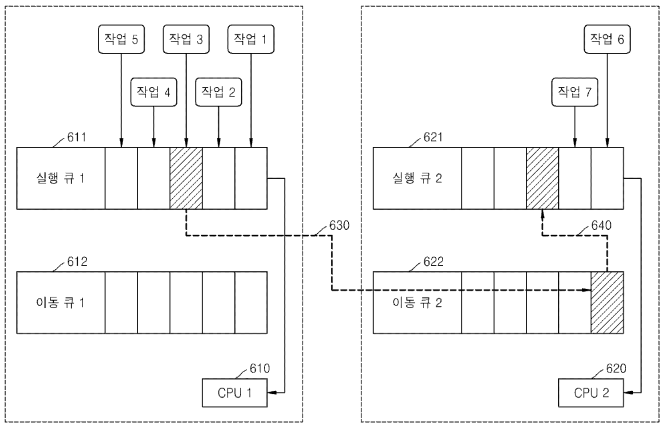
전체 청구항 수 : 총 15 항

(54) 대칭적 다중 프로세서 시스템에서의 로드 밸런싱 방법 및장치

(57) 요 약

본 발명은 대칭적 다중 프로세서 시스템에서의 로드 밸런싱(load balancing) 방법 및 장치에 관한 것으로, 본 발명에 따른 로드 밸런싱 방법은 복수 개의 프로세서들 간의 로드를 고려하여 적어도 두 개의 프로세서들을 선택하는 단계, 선택된 프로세서들 중 제 1 프로세서의 실행 큐(run queue)에 저장된 소정의 작업을 제 2 프로세서의 이동 큐(migration queue)로 이동시키는 단계 및 제 2 프로세서의 이동 큐에 저장된 작업을 제 2 프로세서의 실행 큐로 이동시킴으로써, 작업 이동에 의해 프로세서의 실행 큐가 차단(block)되지 않고, 즉각적인 응답이 가능하며, 스케줄러의 대기 시간이 감소하므로 스케줄러는 컨텍스트 스위칭(context switching)을 더 빨리 실행할 수 있게 되어 전체 운영체제의 성능이 향상된다.

대 표 도



특허청구의 범위

청구항 1

(a) 복수 개의 프로세서들 간의 로드(load)에 기초하여 적어도 두 개의 프로세서들을 선택하는 단계;

(b) 상기 선택된 프로세서들 중 제 1 프로세서의 실행 큐(run queue)에 저장된 소정의 작업을 제 2 프로세서의 이동 큐(migration queue)로 이동시키는 단계; 및

(c) 상기 제 2 프로세서의 이동 큐에 저장된 작업을 상기 제 2 프로세서의 실행 큐로 이동시키는 단계를 포함하는 것을 특징으로 하는 로드 밸런싱(load balancing) 방법.

청구항 2

제 1 항에 있어서,

(d) 상기 제 2 프로세서의 실행 큐에 대한 락(lock)을 획득하는 단계를 더 포함하고,

상기 (c) 단계는 상기 (d) 단계에서의 락 획득이 완료되면, 상기 제 2 프로세서의 이동 큐에 저장된 작업 모두를 상기 제 2 프로세서의 실행 큐로 이동시키고,

(e) 상기 (c) 단계에서의 이동이 완료되면, 상기 제 2 프로세서의 실행 큐에 대한 락을 해제(release)하는 단계를 더 포함하는 것을 특징으로 하는 로드 밸런싱 방법.

청구항 3

제 2 항에 있어서,

(f) 상기 제 2 프로세서의 이동 큐에 대한 락을 획득하는 단계를 더 포함하고,

상기 (c) 단계는 상기 (d) 단계 및 상기 (f) 단계에서의 락 획득이 완료되면, 상기 제 2 프로세서의 이동 큐에 저장된 작업 모두를 상기 제 2 프로세서의 실행 큐로 이동시키고,

(g) 상기 (c) 단계에서의 작업 이동이 완료되면, 상기 제 2 프로세서의 이동 큐에 대한 락을 해제하는 단계를 더 포함하는 것을 특징으로 하는 로드 밸런싱 방법.

청구항 4

제 1 항에 있어서,

(h) 상기 제 2 프로세서의 이동 큐에 대한 락을 획득하는 단계를 더 포함하고,

상기 (b) 단계는 상기 (h) 단계에서의 락 획득이 완료되면, 상기 제 1 프로세서의 실행 큐에 저장된 소정의 작업을 상기 제 2 프로세서의 이동 큐로 이동시키고,

(i) 상기 (b) 단계에서의 작업 이동이 완료되면, 상기 제 2 프로세서의 이동 큐에 대한 락을 해제하는 단계를 더 포함하는 것을 특징으로 하는 로드 밸런싱 방법.

청구항 5

제 4 항에 있어서,

(j) 상기 제 1 프로세서의 실행 큐에 대한 락을 획득하는 단계를 더 포함하고,

상기 (b) 단계는 상기 (h) 단계 및 상기 (j) 단계에서의 락 획득이 완료되면, 상기 제 1 프로세서의 실행 큐에 저장된 소정의 작업을 상기 제 2 프로세서의 이동 큐로 이동시키고,

(k) 상기 (b) 단계에서의 작업 이동이 완료되면, 상기 제 1 프로세서의 실행 큐에 대한 락을 해제하는 단계를 더 포함하는 것을 특징으로 하는 로드 밸런싱 방법.

청구항 6

제 1 항에 있어서,

상기 제 2 프로세서의 이동 큐에 저장된 작업이 있는지 여부를 검사하는 단계를 더 포함하고,

상기 (c) 단계는 상기 검사 결과에 따라 선택적으로 상기 제 2 프로세서의 이동 큐에 저장된 작업을 상기 제 2 프로세서의 실행 큐로 이동시키는 것을 특징으로 하는 로드 밸런싱 방법.

청구항 7

제 1 항에 있어서,

상기 제 2 프로세서의 이동 큐에 대한 락을 획득할 수 있는지 여부를 검사하는 단계를 더 포함하고,

상기 (c) 단계는 상기 검사 결과에 따라 선택적으로 상기 제 2 프로세서의 이동 큐에 저장된 작업을 상기 제 2 프로세서의 실행 큐로 이동시키는 것을 특징으로 하는 로드 밸런싱 방법.

청구항 8

제 1 항 내지 제 7 항 중에 어느 한 항의 방법을 컴퓨터에서 실행시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

청구항 9

복수 개의 프로세서들;

상기 프로세서들의 로드에 기초하여 적어도 두 개의 프로세서들을 선택하는 스케줄러;

상기 선택된 프로세서들 중 제 1 프로세서가 실행할 작업들을 저장하는 상기 제 1 프로세서의 실행 큐;

상기 선택된 프로세서들 중 제 2 프로세서가 실행할 작업들을 저장하는 상기 제 2 프로세서의 실행 큐; 및

상기 제 2 프로세서가 아닌 다른 프로세서의 실행 큐로부터 이동된 작업을 저장하는 상기 제 2 프로세서의 이동 큐를 포함하고,

상기 스케줄러는 상기 제 1 프로세서의 실행 큐에 저장된 소정의 작업을 상기 제 2 프로세서의 이동 큐에 이동시키고, 상기 제 2 프로세서의 이동 큐에 저장된 작업을 상기 제 2 프로세서의 실행 큐로 이동시키는 것을 특징으로 하는 대칭적 다중 프로세서 시스템.

청구항 10

제 9 항에 있어서,

상기 스케줄러는 상기 제 2 프로세서의 실행 큐에 대한 락을 획득하고, 상기 실행 큐에 대한 락 획득이 완료되면 상기 제 2 프로세서의 이동 큐에 저장된 작업 모두를 상기 제 2 프로세서의 실행 큐로 이동시키며, 상기 제 2 프로세서의 이동 큐로부터 상기 제 2 프로세서의 실행 큐로의 작업 이동이 완료되면 상기 제 2 프로세서의 실행 큐에 대한 락을 해제하는 것을 특징으로 하는 대칭적 다중 프로세서 시스템.

청구항 11

제 10 항에 있어서,

상기 스케줄러는 상기 제 2 프로세서의 이동 큐에 대한 락을 획득하고, 상기 실행 큐 및 이동 큐에 대한 락 획득이 완료되면 상기 제 2 프로세서의 이동 큐에 저장된 작업 모두를 상기 제 2 프로세서의 실행 큐로 이동시키며, 상기 제 2 프로세서의 이동 큐로부터 상기 제 2 프로세서의 실행 큐로의 작업 이동이 완료되면 상기 제 2 프로세서의 이동 큐에 대한 락을 해제하는 것을 특징으로 하는 대칭적 다중 프로세서 시스템.

청구항 12

제 9 항에 있어서,

상기 스케줄러는 상기 제 2 프로세서의 이동 큐에 대한 락을 획득하고, 상기 이동 큐에 대한 락 획득이 완료되면 상기 제 1 프로세서의 실행 큐에 저장된 소정의 작업을 상기 제 2 프로세서의 이동 큐로 이동시키며, 상기 제 1 프로세서의 실행 큐로부터 상기 제 2 프로세서의 이동 큐로의 작업 이동이 완료되면 상기 제 2 프로세서의

이동 큐에 대한 락을 해제하는 것을 특징으로 하는 대칭적 다중 프로세서 시스템.

청구항 13

제 12 항에 있어서,

상기 스케줄러는 상기 제 1 프로세서의 실행 큐에 대한 락을 획득하고, 상기 실행 큐 및 이동 큐에 대한 락 획득이 완료되면 상기 제 1 프로세서의 실행 큐에 저장된 소정의 작업을 상기 제 2 프로세서의 이동 큐로 이동시키며, 상기 제 1 프로세서의 실행 큐로부터 상기 제 2 프로세서의 이동 큐로의 작업 이동이 완료되면 상기 제 1 프로세서의 실행 큐에 대한 락을 해제하는 것을 특징으로 하는 대칭적 다중 프로세서 시스템.

청구항 14

제 9 항에 있어서,

상기 스케줄러는 상기 제 2 프로세서의 이동 큐에 저장된 작업이 있는지 여부를 검사하고, 상기 검사 결과에 따라 선택적으로 상기 제 2 프로세서의 이동 큐에 저장된 작업을 상기 제 2 프로세서의 실행 큐로 이동시키는 것을 특징으로 하는 대칭적 다중 프로세서 시스템.

청구항 15

제 9 항에 있어서,

상기 스케줄러는 상기 제 2 프로세서의 이동 큐에 대한 락을 획득할 수 있는지 여부를 검사하고, 상기 검사 결과에 따라 선택적으로 상기 제 2 프로세서의 이동 큐에 저장된 작업을 상기 제 2 프로세서의 실행 큐로 이동시키는 것을 특징으로 하는 대칭적 다중 프로세서 시스템.

명 세 서

발명의 상세한 설명

기 술 분 야

<1> 본 발명은 대칭적 다중 프로세서 시스템(Symmetric Multi-Processor System)에서 운영체제(Operating System)가 로드 밸런싱(load balancing)위하여 작업들(tasks)을 현재의 프로세서에서 다른 프로세서로 이동(migration)시키는 방법 및 장치에 관한 것이다.

배 경 기 술

<2> 현재의 프로세서 아키텍처(processor architecture)의 발전 방향은 하나의 칩(chip)에 다수의 프로세서를 탑재함으로써, 동시에 다수의 작업을 실행할 수 있게 되었다. 그러나, 프로세서의 개수가 증가하는 만큼 시스템의 성능(performance)이 비례하여 향상되지 못하는 현상이 발생하였다. 그 이유는 모든 프로세서들이 균등하게 작업들을 분배하여 실행하는 것이 아니라 다른 프로세서에 비해 특정 프로세서에 더 많은 작업이 할당되어 실행하는 로드 불균형(load imbalance)이 발생하기 때문이다. 따라서, 이러한 문제점을 해결하기 위해 로드가 많은 프로세서에서 상대적으로 로드가 적은 다른 프로세서로 작업을 이동시켜 각 프로세서에 할당되는 작업들의 균형을 맞추는 로드 밸런싱이 필요하다.

<3> 도 1은 종래의 대칭적 다중 프로세서 시스템에서 로드 밸런싱 장치를 도시한 도면으로, 편의상 2개의 CPU(110, 120)만을 도시하였다. CPU들(110, 120)은 각각의 CPU에 대응하여 실행할 작업들이 저장되는 실행 큐들(run queues)(111, 121)을 가진다. 상기의 구성에서 로드 밸런싱 과정은 5개의 작업을 가지고 있어 로드가 많은 CPU 1의 실행 큐 1(111)로부터 2개의 작업을 가지고 있어 상대적으로 로드가 적은 CPU 2의 실행 큐 2(121)로 작업을 이동시킴(130)으로써 달성된다. 이러한 이동 과정(130)에서 우선 현재 작업이 저장되어 있는 실행 큐 1(111)에 대한 락(lock)을 획득하고, 그 다음으로 이동해야할 대상 실행 큐 2(121)에 대한 락을 획득한다.

<4> 2개의 실행 큐들(111, 121)에 대해 락을 획득하고 작업을 이동시키는 이유는 2개의 실행 큐들에 대해 락을 획득하지 않으면 이동 중에 실행 큐에 대한 작업을 추가하거나 삭제하는 경우에 있어서 실행 큐의 무결성(integrity)이 깨질 수 있기 때문이다. 실행 큐의 무결성이 깨지면 특정한 작업이 실행 큐로부터 제외되어 그 특정한 작업이 영원히 스케줄링되지 않을 수 있으며 실행 큐의 구조가 손상되어(corrupted) 운영체제가 깨어지

는(crash) 현상이 발생할 수 있다. 이와 같은 실행 큐들은 뮤텍스(mutex, mutual exclusion object) 또는 세마포어(semaphore)와 같이 다수의 작업들이 동시에 하나의 자원을 사용하는 것을 배제하는 상호 배제의 성질을 갖는 객체들이다. 뮤텍스와 같이 하나의 작업만이 사용할 수 있는 자원(resource)을 임계 영역(critical section)이라는 용어로 표현하기도 한다. 따라서, 작업을 이동시킨 후에는 각각의 실행 큐들에 대한 락을 해제(release)해야만 다른 작업들이 실행 큐들을 사용할 수 있게 된다. 이처럼 락 획득(acquire)과 락 해제(release)는 항상 쌍(pair)으로 동작되어야 할 것이다. 왜냐하면, 소정 작업에 대한 락을 획득한 후 락 해제가 뒤따르지 않으면 다른 작업이 무한히 기다리게 되는 문제가 발생하기 때문이다. 이렇게 2개의 실행 큐들에 대해 동시에 락을 획득하는 방식으로 인해 작업 실행 시간이 지연되는 문제점이 발생하고 결과적으로 운영체제의 성능에 매우 부정적인 영향을 준다.

<5> 도 2는 종래의 대칭적 다중 프로세서 시스템에서 로드 밸런싱 방법을 도시한 흐름도이다. 우선, 201 단계에서 로드 불균형이 발생했는지 여부를 검사한다. 로드 불균형이 발생한 경우 202 단계에서 로드 불균형이 발생한 제 1 CPU, 제 2 CPU와 이동시킬 작업을 선택한다. 203 단계에서 제 1 CPU의 실행 큐에 대해 락 획득을 시도하고 락이 획득되지 않을 경우 락이 획득될 때까지 대기한다. 락이 획득된 경우 다음 단계로 진행한다. 203 단계와 마찬가지로 204 단계에서 제 2 CPU의 실행 큐에 대해 락 획득을 시도한다. 제 1 CPU와 제 2 CPU의 실행 큐에 대한 락을 획득하였으면 205 단계에서 제 1 CPU의 실행 큐로부터 제 2 CPU의 실행 큐로 소정의 작업을 이동시킨다. 206 단계와 207 단계에서 제 1 CPU와 제 2 CPU에 대한 락을 해제한다.

<6> 도 3은 종래의 대칭적 다중 프로세서 시스템에서 로드 밸런싱에 관한 스케줄링 방법을 도시한 흐름도이다. 301 단계에서 스케줄러는 제 2 CPU의 실행 큐에 대해 락 획득을 시도하고 락이 획득되지 않을 경우 락이 획득될 때까지 대기한다. 락이 획득된 경우 302 단계에서 다음에 실행할 작업을 스케줄링 정책(scheduling policy)에 의해 선택한다. 303 단계에서 제 2 CPU의 실행 큐에 대한 락을 해제하고, 304 단계에서 현재 CPU가 실행중인 작업을 선택된 작업으로 교환하는 컨텍스트 스위칭(context switching)을 수행한다.

<7> 이상에서 도 2와 도 3을 통해 종래의 대칭적 다중 프로세서 시스템에서의 로드 밸런싱 방법을 살펴보았다. 이하에서는 이러한 종래의 로드 밸런싱 방법이 어떤 점에서 문제가 되는지를 도 4를 통해 설명하겠다.

<8> 도 4는 종래의 대칭적 다중 프로세서 시스템에서 작업 이동시 시간에 따른 작업들의 실행 순서를 도시한 도면으로, 이상에서 설명한 락으로 인한 스케줄링의 지연 현상을 보여준다. 도 4와 관련하여 최초에 작업 1(410), 작업 2(420) 및 작업 3(430)이 각각 CPU 1, CPU 2 및 CPU 3에 할당되고, 작업 3(430)이 가장 로드가 작은 경우를 가정하자. 상기 가정과 같이 로드 불균형이 발생한 경우, 스케줄러가 작업 1(410)을 CPU 1에서 CPU 3으로 이동시키기로 결정한다. 우선, 작업 1(410)이 저장되어 있는 CPU 1의 실행 큐에 대한 락을 획득하고, 다음으로 이동할 대상 CPU 3의 실행 큐에 대한 락을 획득한다. 이 때, 스케줄러가 작업 2(420)를 CPU 2에서 CPU 3으로 이동시키기로 결정한다. 작업 1(410)의 경우와 마찬가지로 작업 2(420)가 저장되어 있는 CPU 2의 실행 큐에 대한 락을 획득하고, 다음으로 이동할 대상 CPU 3의 실행 큐에 대한 락을 획득한다. 그런데, 작업 2(420)에 의해 상기 CPU 3의 실행 큐에 대한 락 획득이 요청된 시점에 이미 작업 CPU 3의 실행 큐는 1(410)에 의해 락이 획득된 상태이다. 따라서, CPU 3의 실행 큐에 대한 락이 해제(release)될 때까지 계속 대기(440)해야만 하고, 상기 CPU 3의 실행 큐는 작업 1을 위한 락이 해제된 이후에 작업 2를 위해 락이 획득될 수 있다.

<9> 한편, CPU 3의 실행 큐에 대해 스케줄러가 스케줄링을 할 경우, 현재 작업 1과 작업 2를 위해 CPU 3의 실행 큐에 대한 락이 획득되어 있으므로, 상기 CPU 3의 실행 큐에 대한 락이 해제될 때까지 대기(450)해야만 하고, 작업 2를 위한 락이 해제된 이후에 상기 CPU 3의 실행 큐에 대한 락이 획득되고, 스케줄링 작업을 수행한다. 따라서, 종래의 다중 프로세서 시스템의 로드 밸런싱 방법에 있어서 실행 큐를 선점하기 위한 경합(즉, 실행 큐에 대한 락을 획득하기 위한 경합)으로 인해 스케줄러가 장시간 대기하는 문제점이 있었고, 이로 인해 전체 시스템의 성능이 저하되고, 각 작업들의 응답 시간(response time)이 현저하게 지연되는 원인이 되었다. 특히, 이러한 응답 시간 지연 현상은 임베디드 시스템(Embedded System)과 같은 실시간운영체제(Real-Time Operating System)에서 즉각적인 응답 보장에 매우 부정적인 영향을 끼친다.

발명의 내용

해결 하고자하는 과제

<10> 본 발명이 이루고자 하는 기술적 과제는 다중 프로세서 시스템에서 로드 불균형 발생시의 작업 이동에 의해 프로세서의 실행 큐가 차단되어 각 작업들의 응답 시간이 지연되는 현상을 막고, 이로 인해 스케줄러가 스케줄링을 수행하지 못하고 장시간 대기하게 되며, 결과적으로 전체 운영체제의 성능이 나빠지는 문제점을 해결하는

로드 밸런싱 방법 및 장치를 제공하는데 있다.

과제 해결수단

<11> 상기 기술적 과제를 달성하기 위하여, 본 발명에 따른 로드 밸런싱(load balancing) 방법은 복수 개의 프로세서들 간의 로드(load)에 기초하여 적어도 두 개의 프로세서들을 선택하는 단계; 상기 선택된 프로세서들 중 제 1 프로세서의 실행 큐(run queue)에 저장된 소정의 작업을 제 2 프로세서의 이동 큐(migration queue)로 이동시키는 단계; 및 상기 제 2 프로세서의 이동 큐에 저장된 작업을 상기 제 2 프로세서의 실행 큐로 이동시키는 단계를 포함하는 것을 특징으로 한다.

<12> 상기 다른 기술적 과제를 해결하기 위하여, 본 발명은 상기 기재된 로드 밸런싱 방법을 컴퓨터에서 실행시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체를 제공한다.

<13> 상기 기술적 과제를 달성하기 위하여, 본 발명에 따른 다중 프로세서 시스템은 복수 개의 프로세서들; 상기 프로세서들의 로드에 기초하여 적어도 두 개의 프로세서들을 선택하는 스케줄러; 상기 선택된 프로세서들 중 제 1 프로세서가 실행할 작업들을 저장하는 상기 제 1 프로세서의 실행 큐; 상기 선택된 프로세서들 중 제 2 프로세서가 실행할 작업들을 저장하는 상기 제 2 프로세서의 실행 큐; 및 상기 제 2 프로세서가 아닌 다른 프로세서의 실행 큐로부터 이동된 작업을 저장하는 상기 제 2 프로세서의 이동 큐를 포함하고, 상기 스케줄러는 상기 제 1 프로세서의 실행 큐에 저장된 소정의 작업을 상기 제 2 프로세서의 이동 큐에 이동시키고, 상기 제 2 프로세서의 이동 큐에 저장된 작업을 상기 제 2 프로세서의 실행 큐로 이동시키는 것을 특징으로 한다.

효 과

<14> 본 발명은 종래 대칭적 다중 프로세서 시스템에서 로드 불균형 발생시의 작업 이동에 의해 프로세서의 실행 큐가 차단되어 각 작업들의 응답 시간이 지연되는 현상을 막고, 이로 인해 스케줄러가 스케줄링을 수행하지 못하고 장시간 대기하게 되는 문제점을 해결하기 위해, 프로세서들 간의 소정 작업의 이동시 실행 큐가 아닌 이동 큐로 소정 작업을 이동시키고, 이러한 이동 큐가 다른 작업에 의해 차단되지 않는 스케줄링 때에 이동 큐로부터 이동 큐에 대응하는 실행 큐로 소정 작업을 이동시킴으로써, 작업 이동에 의해 프로세서의 실행 큐가 차단되지 않고, 즉각적인 응답이 가능한 로드 밸런싱 방법 및 장치를 제공한다. 따라서 본 발명에 의해 스케줄러가 대기하는 시간이 감소하고, 스케줄러는 컨텍스트 스위칭을 더 빨리 실행할 수 있게 되므로 전체 운영체제의 오버헤드(overhead)가 줄어들며, 작업 간의 빠른 전환으로 인해 임베디드 시스템에서의 실시간성 보장이 강화된다.

발명의 실시를 위한 구체적인 내용

<15> 이하에서는 도면을 참조하여 본 발명의 바람직한 실시예들을 상세히 설명한다.

<16> 도 5는 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템의 구성도이다.

<17> 도 5를 참조하면, 본 실시예에 따른 다중 프로세서 시스템은 다중 프로세서(510), 메모리(520), 스케줄러(530) 및 버스(540)로 구성된다. 다만, 도 5에 도시된 다중 프로세서 시스템은 본 실시예를 설명하기 위한 가장 간단한 일례에 불과하며, 상기 구성 외에 디스크 장치 및 입출력 장치 등의 다른 구성 요소들이 포함될 수 있음을 본 실시예가 속하는 기술 분야에서 통상의 지식을 가진 자라면 이해할 수 있다. 또한, 도 5는 각각의 구성의 역할에 따라 도시한 관념적인 구성도이므로 물리적인 하드웨어 구성과는 다소 차이가 있을 수 있다.

<18> 다중 프로세서(510)는 메모리에 저장된 작업들을 실행하는 복수 개의 CPU들로 구성된다. 메모리(520)는 CPU가 작업을 처리하는 작업공간으로서 운영체제의 각종 프로그램과 작업들이 적재된다. 또한, 메모리(520)는 CPU들 각각에 대응하여 CPU들이 실행할 작업들이 저장되는 실행 큐들을 가지며, 본 실시예에서는 CPU 1개당 실행 큐 1개를 대응시키도록 예시하고 있다. 더불어, 메모리에는 실행 큐들 이외에 각각의 실행 큐에 대응하는 이동 큐들을 갖는다. 이동 큐는 종래의 로드 밸런싱에서 소스 실행 큐(source run queue)에서 대상 실행 큐(target run queue)로 직접 작업이 이동하였던 것과는 달리, 소스 실행 큐(source run queue)로부터 이동(migration)하는 작업을 저장하는 큐(queue)이다. 즉, 이동 큐는 자신이 아닌 다른 CPU의 실행 큐로부터 작업을 이동시켜 저장한다. 예를 들어, 도 5에 도시된 CPU 1이 실행 큐 1과 이동 큐 1에 대응되고, 마찬가지로 CPU 2가 실행 큐 2와 이동 큐 2에 대응된다면, 이동 큐 1은 CPU 1이 아닌 CPU 2나 CPU 3의 실행 큐로부터 작업을 이동시켜 저장할 수 있다.

<19> 스케줄러(530)는 다중 프로세서 시스템의 다양한 작업들을 스케줄링하여 필요한 자원에 공급한다. 일반적으로

스케줄러에는 메모리 내의 실행 큐들에 저장된 작업들을 소정의 스케줄링 정책에 따라 각각의 대응하는 CPU에 작업을 할당하는 CPU 스케줄러와 디스크 장치(미도시)와 같은 저장 장치로부터 메모리로 필요한 프로그램을 적재하는 작업 스케줄러 등이 있으며, 본 실시예에서는 CPU 스케줄러를 중심으로 기술하고 있으므로, 이하에서는 편의상 스케줄러라고 줄여 호칭하겠다. 본 발명의 바람직한 일 실시예에 따른 스케줄러(530)는 다중 프로세서 시스템에서 로드 불균형이 발생한 경우 소정 CPU의 실행 큐로부터 다른 CPU의 이동 큐로 시스템 버스(540)를 통해 작업을 이동시킨다. 또한, 실행 큐들에 저장된 작업들을 소정의 스케줄링 정책에 따라 선택하여 CPU에 할당한다. 소정의 스케줄링 정책은 우선순위에 따른 스케줄링 정책 등 통상의 운영체제에서 지원하는 다양한 스케줄링 방법이 알려져 있으므로 자세한 설명은 생략한다.

<20> 일반적으로 운영체제 내에서 CPU에 작업을 할당하거나 실행 큐들에 대하여 락을 획득하고, 락을 해제하고, 작업을 이동시키는 것은 모두 스케줄러(530)에 의해 이루어지는 것으로 본 발명에 속하는 기술 분야에서 통상의 지식을 가진 자는 이하의 실시예들에서 설명할 각각의 단계들에서 행위 주체를 특별히 언급하지 않더라도 스케줄러(530)에 의해 이루어지는 것임은 알 수 있다. 또한, 이동 큐 역시 메모리(520)에서 관리되는 큐의 일종이므로 스케줄러(530)에 의해 락이 획득되고, 락이 해제되며, 작업을 저장하거나 이동시키게 된다.

<21> 도 6은 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템에서 이동 큐를 이용한 로드 밸런싱 장치를 도시한 도면으로, 편의상 2개의 CPU(610, 620)만을 도시하였다. CPU들(610, 620)은 각각의 CPU들에 대응하여 실행할 작업이 저장되는 실행 큐들(611, 621)을 가지고, 또한 각각의 실행 큐들에 대응하는 이동 큐들(612, 622)을 가진다.

<22> 도 6에 도시된 구성에서 로드 밸런싱을 수행하는 과정은 5개의 작업을 가지고 있어 로드가 많은 CPU 1의 실행 큐 1(611)로부터 2개의 작업을 가지고 있어 상대적으로 로드가 적은 CPU 2의 이동 큐 2(622)로 작업을 이동시킴(630)으로써 달성된다. 여기서 실행 큐들(611, 621)과 이동 큐들(612, 622)은 도 1에서 설명한 뮤텍스와 같이 다수의 작업들이 동시에 하나의 자원을 사용하는 것을 배제하는 상호 배제의 성질을 갖는 객체들인 것이 바람직하다. 따라서, 도 1의 경우와 유사하게 작업 이동에 관여하는 출발점과 도착점의 2개의 큐 모두에 대한 락을 획득하고 락을 해제하는 과정이 필요하다. 즉, 소정 작업을 이동시키기 전에 2개의 큐들 모두에 대해 락을 획득하고, 소정 작업을 이동시킨 후에 2개의 큐들 모두에 대한 락을 해제함으로써 2개의 큐들에 대한 무결성을 보장할 수 있게 된다. 여기서 락은 하나의 프로세스(process)가 큐들을 사용할 때 다른 프로세스가 이러한 큐들을 사용하지 못하는 배타적 권한을 의미한다.

<23> 이러한 락 획득과 락 해제의 과정을 도 3을 통해 구체적으로 살펴보면, 우선 CPU 1(610)의 실행 큐 1(611)과 CPU 2(620)의 이동 큐 2(622)에 대한 락을 획득하고, 그 다음에 CPU 1의 실행 큐 1(611)로부터 CPU 2의 이동 큐 2(622)로 작업 3을 이동시키고(630), 마지막으로 CPU 1의 실행 큐 1(611)과 CPU 2의 이동 큐 2(622)에 대한 락을 해제하게 된다.

<24> 이상의 과정에서 CPU 2의 이동 큐 2(622)에 저장된 작업 3은 이동만 되었을 뿐, 직접 CPU 2(620)에 부여되어 실행되지는 않는다. 이동 큐 2(622) 저장된 작업 3은 이동 큐 2에 대응하는 실행 큐 2(621)에 전달되어야만 비로소 스케줄러에 의한 스케줄링이 가능해진다. 따라서, 이동 큐 2(622)에 저장된 작업 3을 실행 큐 2(621)로 이동시키는(640) 과정이 필요하며, 이러한 큐들(622, 621)도 상기 CPU 1의 실행 큐 1(611)로부터 CPU 2의 이동 큐 2(622)로 이동시키는(630) 과정과 마찬가지로 락 획득과 락 해제 과정에 따를 것이다. 구체적으로, 우선 CPU 2의 이동 큐 2(622)와 CPU 2의 실행 큐 2(621)에 대한 락을 획득하고, 그 다음에 CPU 2의 이동 큐 2(622)로부터 CPU 2의 실행 큐 2(621)로 저장되어 있던 작업 3을 이동시키고(640), 마지막으로 CPU 2의 이동 큐 2(622)와 CPU 2의 실행 큐 2(621)에 대한 락을 해제하게 된다.

<25> 한편, 상기 CPU 2의 이동 큐 2(622)로부터 CPU 2의 실행 큐 2(621)로 작업을 이동시키는 것(640)과 같이 동일 CPU 내의 이동 큐에서 실행 큐로 작업을 이동시키는 과정이 실행되고 있을 때, 스케줄러가 CPU 2(620)에서 작업을 실행시키기 위해 CPU 2의 실행 큐 2(621)에 대해 스케줄링을 수행하려 할 경우에 상호 배제의 문제를 조정할 필요가 있다. 즉, CPU 2의 실행 큐 2(621)에 대하여 작업 3을 이동시키려는 스케줄링 프로세스와 CPU 2(620)에 작업을 할당하려는 스케줄링 프로세스가 자원(CPU 2의 실행 큐 2)을 점유하기 위한 경합을 벌이는 경우에 대기 및 지연을 조절하는 것이 문제가 된다.

<26> 이러한 문제를 해결하기 위해 CPU 2의 이동 큐 2(622)로부터 CPU 2의 실행 큐 2(621)로 작업을 이동시키기 전에 스케줄러가 CPU 2의 이동 큐 2(622)에 대한 사용 권한을 획득할 수 있는지 여부를 검사하는 것이 바람직하다. 검사 결과 스케줄러가 CPU 2의 이동 큐 2(622)에 대한 사용 권한을 획득할 수 없다면 CPU 2의 이동 큐 2(622)가 다른 스케줄링 프로세스나 작업 등에 의해 사용되지 않을 때까지 대기할 수도 있고, 대기하지 않을 수도 있다.

<27> 대기하지 않는 경우 본 실시예의 스케줄러는 CPU 2의 이동 큐 2(622)가 사용되지 않는 다음번의 스케줄링 때에 CPU 2의 이동 큐 2(622)로부터 CPU 2의 실행 큐 2(621)로 작업을 이동시킨다. 즉, 스케줄러는 CPU 2의 이동 큐 2(622)로부터 CPU 2의 실행 큐 2(621)로 작업을 이동시키지 않고 CPU 2의 실행 큐 2(621)에 대한 스케줄링만을 수행한다. 여기서 CPU 2의 이동 큐 2(622)가 사용되고 있는지 여부는 CPU 2의 이동 큐 2가 다른 스케줄링 프로세스나 작업 등에 의해 락이 획득되어 있는지 여부를 검사하는 방법으로써 구현될 수 있을 것이다. 스케줄러는 운영체제 내에서 소정의 규칙에 의해 주기적으로 또는 특정 조건을 만족할 경우에 실행되므로, 스케줄러가 CPU 2의 이동 큐 2(622)를 사용할 수 있는 스케줄링 시기에 소정의 작업을 이동시킴으로써 스케줄러가 CPU 2의 이동 큐 2(622)를 사용할 수 있을 때까지 대기하는 것으로 인해 CPU 2의 실행 큐 2(621)에 대한 스케줄링이 지연되는 문제를 막을 수 있다.

<28> 도 7은 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템에서 이동 큐를 이용한 로드 밸런싱 방법을 도시한 흐름도로서, 다음과 같은 단계들로 구성된다.

<29> 701 단계에서 스케줄러는 로드 불균형이 발생했는지 여부를 검사한다. 만약 로드 불균형이 발생하지 않았다면 로드 밸런싱 절차가 종료될 것이고, 그렇지 않다면 702 단계로 진행한다.

<30> 702 단계에서 스케줄러는 제 1 CPU, 제 2 CPU 및 이동시킬 작업을 선택한다. 실행 큐에 저장된 작업이 많아 부하가 큰 CPU에서 저장된 작업이 적어 부하가 작은 CPU로 작업을 분배하기 위해 대상 CPU들과 이동시킬 작업을 선택한다.

<31> 703 단계에서 스케줄러는 제 1 CPU의 실행 큐에 대한 락 획득을 시도한다. 다른 스케줄링 프로세스에 의해 이미 점유되어 락이 획득되지 않는다면 실행 큐에 대한 락 해제를 기다린다. 마찬가지로, 704 단계에서 제 2 CPU의 이동 큐에 대한 락 획득을 시도한다. 락이 획득되지 않는다면 이동 큐의 락 해제를 기다린다. 양 큐들에 대해 모두 락이 획득되었다면 다음 단계로 진행한다.

<32> 705 단계에서 스케줄러는 제 1 CPU의 실행 큐로부터 제 2 CPU의 이동 큐로 소정의 작업을 이동시키고, 706 단계와 707 단계에서 각각 제 2 CPU의 이동 큐와 제 1 CPU의 실행 큐에 대한 락을 해제한다. 이 과정들은 상기 도 6에서 설명한 바와 같으므로 자세한 내용은 생략한다.

<33> 한편, CPU들 간의 작업 이동과 더불어 스케줄러에 의한 스케줄링이 병행될 수 있다. 도 8은 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템에서 이동 큐를 이용한 로드 밸런싱에 관한 스케줄링 방법을 도시한 흐름도로서, 다음과 같은 단계들로 구성된다.

<34> 801 단계에서 스케줄러는 제 2 CPU의 실행 큐에 대한 락 획득을 시도한다. 제 2 CPU의 실행 큐가 이미 다른 스케줄링 프로세스에 의해 점유되어 있어 락이 획득되지 않는다면 실행 큐에 대한 락 해제를 대기하고, 락이 획득되었다면 다음 단계로 진행한다.

<35> 802 단계에서 스케줄러는 제 2 CPU의 이동 큐에 대해 락 획득을 시도한다. 락이 획득되지 않을 경우 락이 해제될 때까지 대기(809)한다. 제 2 CPU의 이동 큐에 대해 락이 획득되었다면 다음 단계로 진행한다.

<36> 한편, 제 2 CPU의 이동 큐에 대한 락이 획득되지 않았을 경우의 대기(809)와 관련하여, 제 2 CPU의 이동 큐로부터 제 2 CPU의 실행 큐로 작업을 이동시키는 과정과 스케줄러가 제 2 CPU에서 작업을 실행시키기 위해 제 2 CPU의 실행 큐에 대해 스케줄링을 수행하는 과정이 경합하는 경우를 가정하자. 이 경우 도 6에서 설명한 바와 같이 제 2 CPU의 이동 큐에 대한 락이 획득되어 있는지 여부를 검사할 수 있다. 만약 제 2 CPU의 이동 큐가 선점되어 있다면 이동 큐에 대한 락이 해제될 때까지 대기(809)할 수도 있고, 도 6에서 설명한 바와 같이 대기하지 않고 제 2 CPU의 실행 큐에 대한 스케줄링을 수행할 수도 있다. 대기하지 않는 경우 다음번의 스케줄러가 실행될 때에 다시 이동 큐에 대한 락 획득 여부를 검사하고, 이동 큐에 대한 락이 획득되어 있지 않을 경우 803 단계로 진행함으로써 CPU 2의 실행 큐에 대한 경합으로 인해 발생할 수 있는 제 2 CPU의 실행 큐에 대한 스케줄링 지연을 피할 수 있다.

<37> 803 단계에서 스케줄러는 제 2 CPU의 이동 큐에 작업이 있는지 여부를 검사한다. 이동 큐에 작업이 없다면 이동 큐로부터 실행 큐로 작업을 이동시킬 필요없이 806 단계로 진행하여 제 2 CPU의 실행 큐에 대한 스케줄링을 수행한다. 만약, 제 2 CPU의 이동 큐에 작업이 있다면 작업 이동을 위한 804 단계로 진행한다.

<38> 804 단계에서 스케줄러는 제 2 CPU의 이동 큐로부터 제 2 CPU의 실행 큐로 소정의 작업을 이동시킨다. 805 단계에서 제 2 CPU의 이동 큐에 대한 락을 해제하고,806 단계에서 다음에 실행할 작업을 선택한 후, 807 단계에서 제 2 CPU의 실행 큐에 대한 락을 해제한다. 마지막으로 808 단계에서 실행 큐에 저장된 작업을 위한 컨텍스트

스위칭을 실행하고 제 2 CPU는 스위칭된 작업을 처리한다.

<39> 도 9는 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템에서 이동 큐를 이용한 작업 이동시 시간에 따른 작업들의 실행 순서를 도시한 도면이다. 도 4와 유사하게 최초에 작업 1(910), 작업 2(920) 및 작업 3(930)이 각각 CPU 1, CPU 2 및 CPU 3에 할당되고, 작업 3(930)이 가장 로드가 작은 경우를 가정하자.

<40> 상기 가정과 같이 로드 불균형이 발생한 경우, 스케줄러는 작업 1(910)을 CPU 1에서 CPU 3으로 이동시키기로 결정한다. 우선, 작업 1(910)이 저장되어 있는 CPU 1의 실행 큐에 대해 락을 획득한다. 다음으로, 도 4의 종래 실시예에서 이동할 대상 CPU 3의 실행 큐에 대해 락을 획득하였던 것과는 달리 본 실시예에서는 CPU 3의 이동 큐에 대해 락을 획득한다. 이 때, 스케줄러가 작업 2(920)를 CPU 2에서 CPU 3으로 이동시키기로 결정한다. 작업 1(910)의 경우와 마찬가지로 작업 2(920)가 저장되어 있는 CPU 2의 실행 큐에 대한 락을 획득하고, 다음으로 이동할 대상 CPU 3의 이동 큐에 대해 락 획득을 시도한다. 그런데, 작업 2(920)에 의해 상기 CPU 3의 이동 큐에 대한 락 획득이 시도된 시점에 이미 CPU 3의 이동 큐는 작업 1(910)에 의해 락이 획득된 상태이다. 따라서, CPU 3의 이동 큐에 대한 락이 해제될 때까지 대기(940)해야만 하고, CPU 3의 이동 큐는 상기 작업 1을 위한 락이 해제된 이후에 작업 2를 위해 락이 획득될 수 있다.

<41> 본 발명의 일 실시예에 따른 다중 프로세서 시스템에서 이동 큐를 이용한 로드 밸런싱 방법은 스케줄러가 CPU 3의 실행 큐에 대해 락 획득을 시도하는 경우와 CPU 1 또는 CPU 2의 실행 큐로부터 CPU 3의 이동 큐로 소정 작업을 이동시키려 하는 경우가 동시에 발생할 때 종래 기술과 큰 차이점이 있다. 도 9에서 CPU 3의 실행 큐에 대해 스케줄러가 스케줄링 1을 수행할 경우, CPU 1과 CPU 2에 할당되어 있는 작업 1(910)과 작업 2(920)가 CPU 3의 이동 큐로 이동하는지 여부와 상관없이 스케줄러는 CPU 3의 실행 큐에 대한 스케줄링 수행이 가능하다. 왜냐하면 CPU 3의 실행 큐는 작업 1 및 작업 2의 이동에 의해 락이 획득되지 않기 때문이다. 즉, 작업 이동 과정에서 CPU 3의 이동 큐만에 대해 락을 획득하기 때문에 스케줄러는 CPU 3의 실행 큐에 대해 락을 획득하고 자유롭게 스케줄링 1을 수행할 수 있다.

<42> 이 과정을 구체적으로 살펴보면, 작업 3(930)이 저장되어 있는 CPU 3의 실행 큐에 대해 스케줄링 1이 시작된다. 스케줄러는 우선 CPU 3의 실행 큐에 대한 락을 획득하고, 다음 실행 작업을 선택한다. 여기서 작업을 실행할 주체는 물론 CPU 3이 될 것이다. 작업이 선택되었으면, CPU 3의 실행 큐에 대한 락을 해제하고, 현재 CPU 3에서 실행중인 작업과 선택된 작업을 교환하는 컨텍스트 스위칭을 실행함으로써, 스케줄링 1이 종료된다. 도 9에서 스케줄러에 의한 스케줄링 1의 과정을 살펴보면, 도 9에는 도 4에서 스케줄러가 CPU 3의 실행 큐에 대한 락이 해제될 때까지 대기하던 구간(450)이 없는 것을 알 수 있을 것이다. 따라서, 이동 큐를 이용한 로드 밸런싱 방법을 통해 스케줄러는 지연없는 스케줄링 실행이 가능하고, 이로 인해 CPU 3은 스케줄링 정책에 따라 선택된 작업을 지연없이 바로 실행할 수 있게 된다.

<43> 한편, CPU 3의 이동 큐에 저장되어 있는 작업 1과 작업 2는 스케줄러에 의해 CPU 3에 할당될 수 없으므로, 스케줄러에 의해 다른 작업들과 더불어 스케줄링 대상이 되는 CPU 3의 실행 큐로 이동시키는 과정이 필요하다. 도 9에서 CPU 3의 이동 큐에 저장되어 있는 작업 1과 작업 2에 대해 CPU 3의 실행 큐로 이동시키려고 할 경우, 우선 CPU 3의 실행 큐에 대한 락을 획득하고, 다음으로 CPU 3의 이동 큐에 대해 락을 획득한다. 이 때, 도 6에서 설명한 경합에 대한 상호 배제의 경우와 같이 CPU 3의 이동 큐에 대한 락이 획득되어 있는지 여부를 검사하는 것이 바람직하다. 도 9에서 스케줄링 2가 시작되기 전에 작업 2를 CPU 2의 실행 큐에서 CPU 3의 이동 큐로 이동하는 과정이 모두 종료되었고, 현재 CPU 3의 이동 큐에 대한 락이 해제된 상태이다. 따라서, 스케줄링 2가 시작되면 CPU 3의 이동 큐에 대해 락이 획득될 것이고, 스케줄러에 의해 CPU 3의 이동 큐 내의 작업들(작업 1 및 작업 2)이 CPU 3의 실행 큐로 이동(950)할 것이다. 이어서, CPU 3의 이동 큐에 대한 락이 해제되고, 다음 실행할 작업을 선택한 다음 CPU 3의 실행 큐에 대한 락이 해제된다. 마지막으로 컨텍스트 스위칭이 실행됨으로써 스케줄링 2가 종료된다.

<44> 도 9에서 스케줄러에 의해 CPU 3의 이동 큐 내의 작업들 모두(작업 1 및 작업 2)가 CPU 3의 실행 큐로 이동(950)하는 것이 바람직하다. 도 4에서 종래 CPU 1의 실행 큐 및 CPU 2의 실행 큐로부터 CPU 3의 실행 큐로 작업들이 이동할 경우 CPU 3의 실행 큐에 대해 각각 1회씩 총 2회의 락이 획득되었음에 반해, 도 9에서는 CPU 3의 이동 큐 내의 작업들 모두(작업 1 및 작업 2)가 CPU 3의 실행 큐로 이동(950)함으로써 단 1회의 락이 획득되었다. 만약 로드 밸런싱을 위해 작업을 이동시킬 CPU가 10개라면 도 4의 종래 실시예에서는 CPU 3의 실행 큐에 대해 총 10회의 락이 획득됨에 반해, 도 9의 실시예에서는 여전히 단 1회의 락만이 획득된다. 따라서, 스케줄러가 CPU 3의 실행 큐에 대한 스케줄링을 위해 필요한 지연 시간이 현저히 줄어든다.

<45> 이상에서 본 발명에 대하여 그 바람직한 실시예들을 중심으로 살펴보았다. 본 발명에 속하는 기술 분야에서 통

상의 지식을 가진 자는 본 발명이 본 발명의 본질적인 특성에서 벗어나지 않는 범위에서 변형된 형태로 구현될 수 있음을 이해할 수 있을 것이다. 그러므로 개시된 실시예들은 한정적인 관점이 아니라 설명적인 관점에서 고려되어야 한다. 본 발명의 범위는 전술한 설명이 아니라 특허청구범위에 나타나 있으며, 그와 동등한 범위 내에 있는 모든 차이점은 본 발명에 포함된 것으로 해석되어야 할 것이다.

도면의 간단한 설명

<46> 도 1은 종래의 대칭적 다중 프로세서 시스템에서 로드 밸런싱 장치를 도시한 도면이다.

<47> 도 2는 종래의 대칭적 다중 프로세서 시스템에서 로드 밸런싱 방법을 도시한 흐름도이다.

<48> 도 3은 종래의 대칭적 다중 프로세서 시스템에서 로드 밸런싱에 관한 스케줄링 방법을 도시한 흐름도이다.

<49> 도 4는 종래의 대칭적 다중 프로세서 시스템에서 작업 이동시 시간에 따른 작업들의 실행 순서를 도시한 도면이다.

<50> 도 5는 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템의 구성도이다.

<51> 도 6은 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템에서 이동 큐를 이용한 로드 밸런싱 장치를 도시한 도면이다.

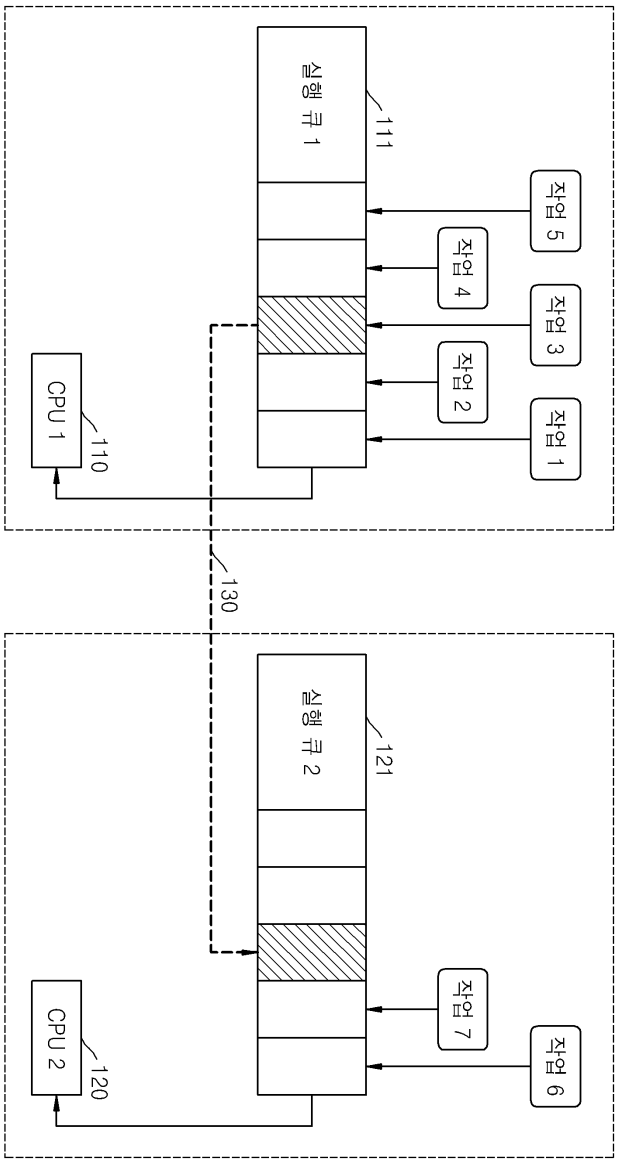
<52> 도 7은 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템에서 이동 큐를 이용한 로드 밸런싱 방법을 도시한 흐름도이다.

<53> 도 8은 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템에서 이동 큐를 이용한 로드 밸런싱에 관한 스케줄링 방법을 도시한 흐름도이다.

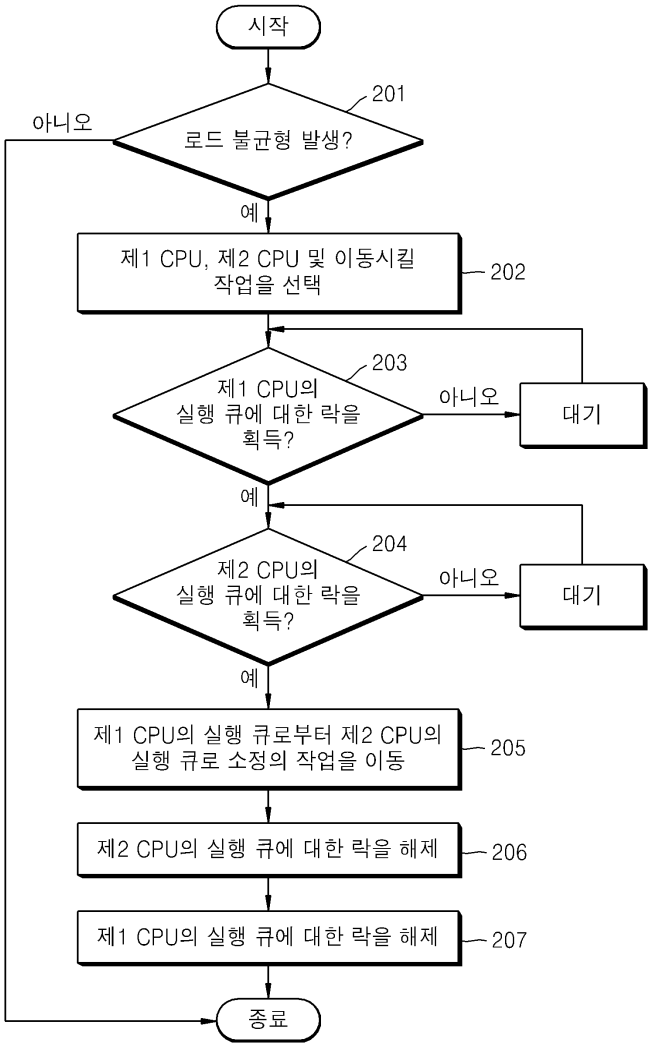
<54> 도 9는 본 발명의 일 실시예에 따른 대칭적 다중 프로세서 시스템에서 이동 큐를 이용한 작업 이동시 시간에 따른 작업들의 실행 순서를 도시한 도면이다.

도면

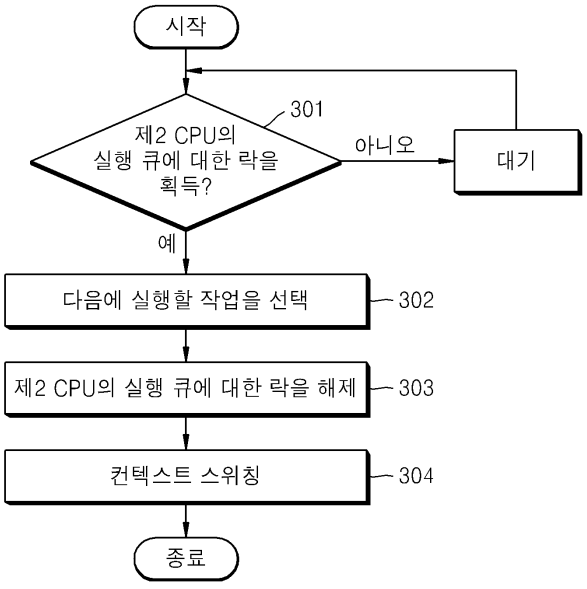
도면1


실행 큐 1
111
작업 5
작업 4
작업 3
작업 2
작업 1
CPU 1
110
130
실행 큐 2
121
작업 7
작업 6
CPU 2
120

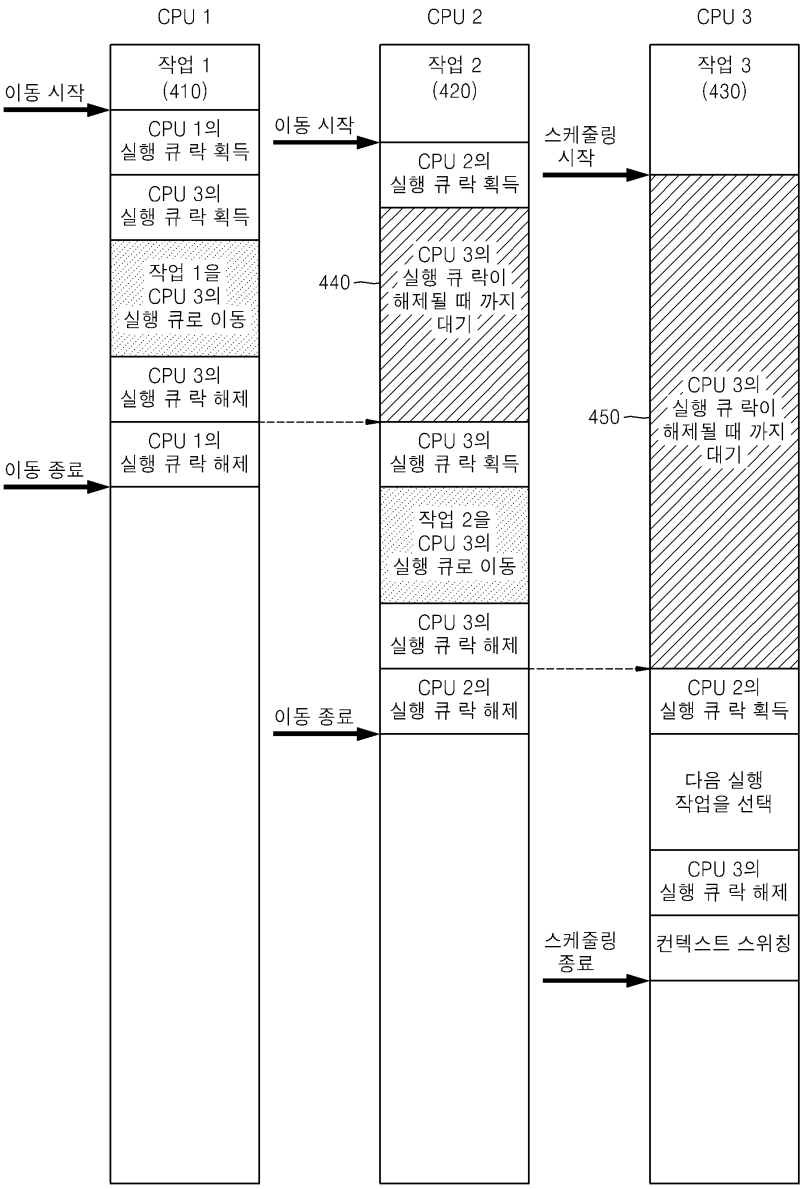
도면2


시작
201
로드 불균형 발생?
아니오
예
제1 CPU, 제2 CPU 및 이동시킬 작업을 선택
202
203
제1 CPU의 실행 큐에 대한 락을 획득?
아니오
대기
예
204
제2 CPU의 실행 큐에 대한 락을 획득?
아니오
대기
예
제1 CPU의 실행 큐로부터 제2 CPU의 실행 큐로 소정의 작업을 이동
205
제2 CPU의 실행 큐에 대한 락을 해제
206
제1 CPU의 실행 큐에 대한 락을 해제
207
종료

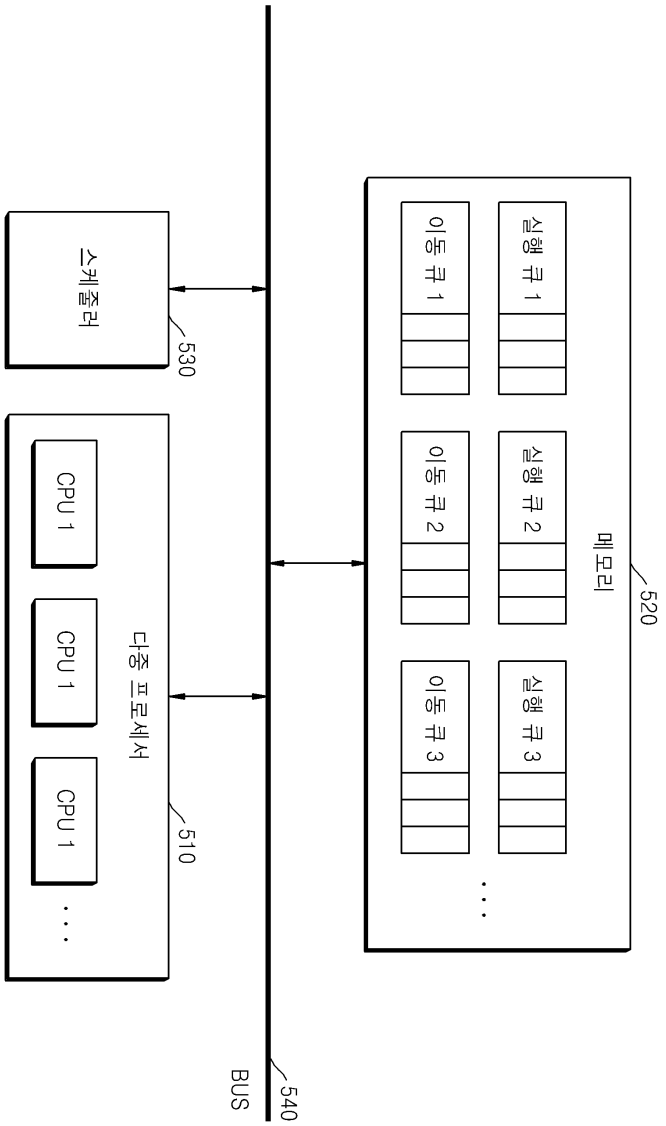
도면3


시작
301
제2 CPU의 실행 큐에 대한 락을 획득?
아니오
대기
예
다음에 실행할 작업을 선택
302
제2 CPU의 실행 큐에 대한 락을 해제
303
컨텍스트 스위칭
304
종료

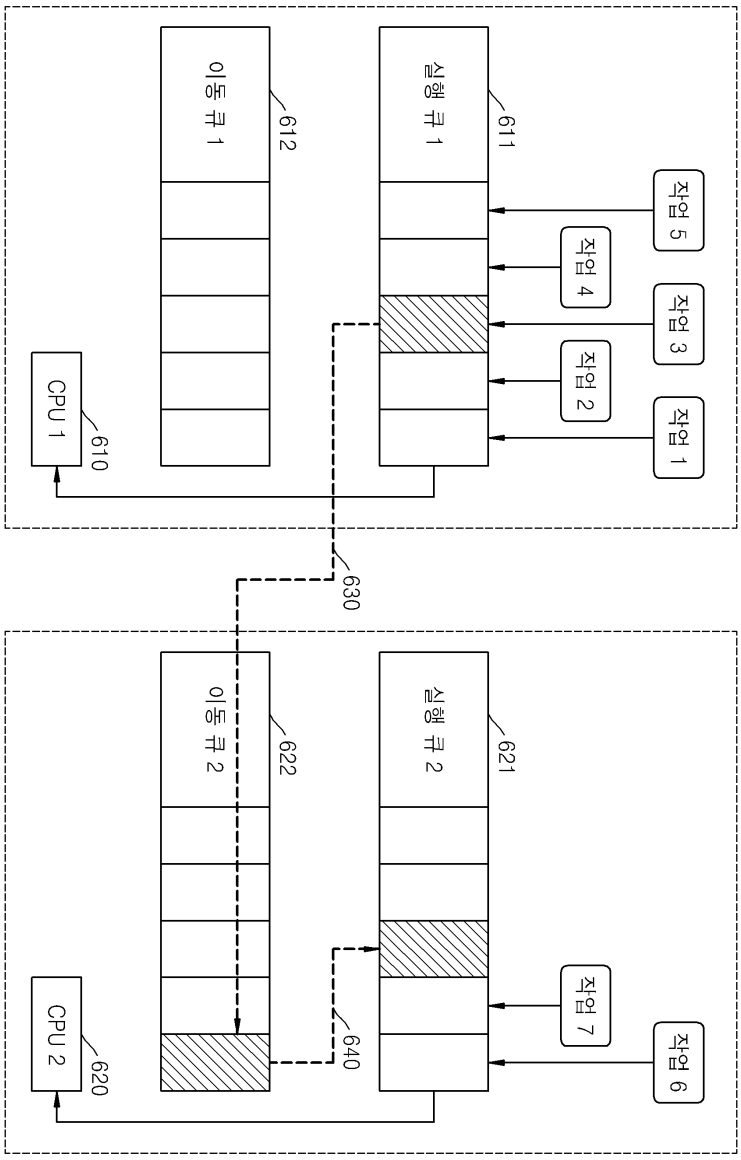
도면4


CPU 1
CPU 2
CPU 3
작업 1
(410)
작업 2
(420)
작업 3
(430)
이동 시작
CPU 1의
실행 큐 락 획득
이동 시작
CPU 2의
실행 큐 락 획득
스케줄링
시작
CPU 3의
실행 큐 락 획득
작업 1을
CPU 3의
실행 큐로 이동
440
CPU 3의
실행 큐 락이
해제될 때 까지
대기
CPU 3의
실행 큐 락 해제
450
CPU 3의
실행 큐 락이
해제될 때 까지
대기
CPU 1의
실행 큐 락 해제
이동 종료
CPU 3의
실행 큐 락 획득
작업 2을
CPU 3의
실행 큐로 이동
CPU 3의
실행 큐 락 해제
CPU 2의
실행 큐 락 해제
이동 종료
CPU 2의
실행 큐 락 획득
다음 실행
작업을 선택
CPU 3의
실행 큐 락 해제
스케줄링
종료
컨텍스트 스위칭

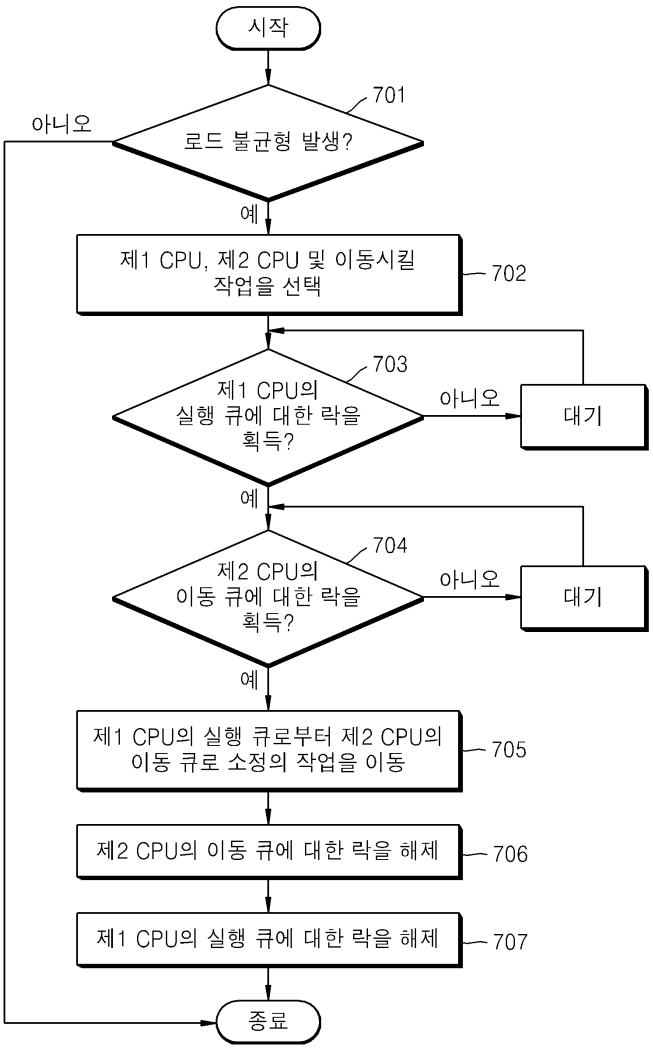
도면5


스케줄러
530
CPU 1
CPU 1
CPU 1
다중 프로세서
510
이동 큐 1
실행 큐 1
이동 큐 2
실행 큐 2
이동 큐 3
실행 큐 3
메모리
520
BUS
540

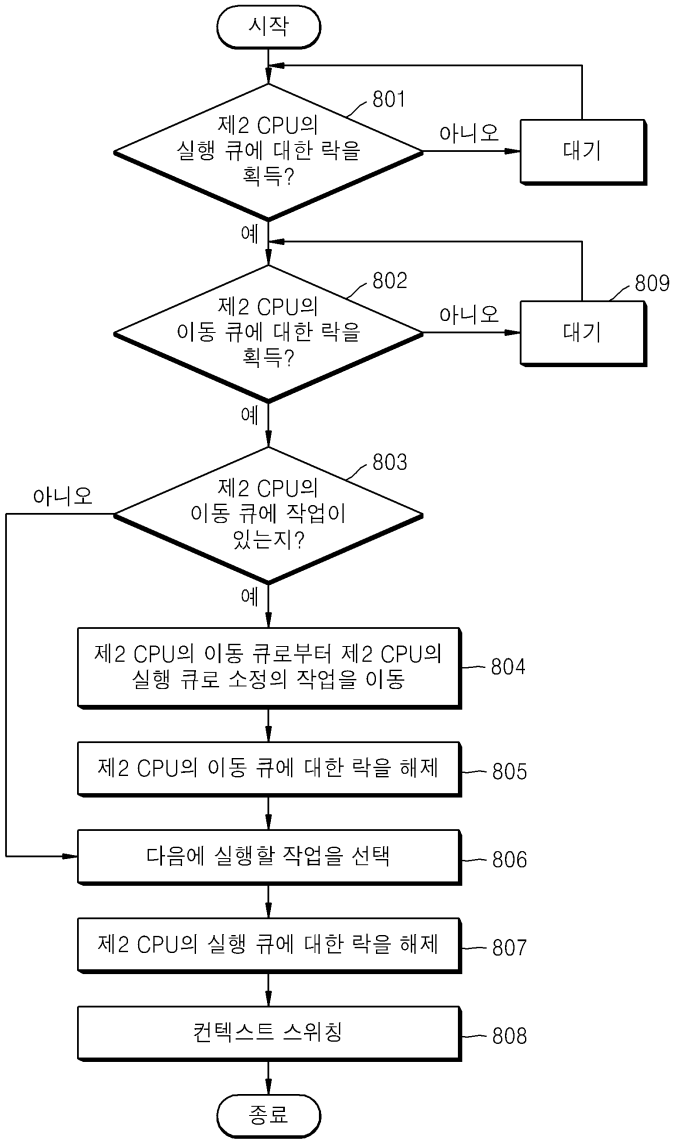
도면6


CPU 1
610
이동 큐 1
612
실행 큐 1
611
작업 5
작업 4
작업 3
작업 2
작업 1
630
CPU 2
620
이동 큐 2
622
실행 큐 2
621
640
작업 7
작업 6

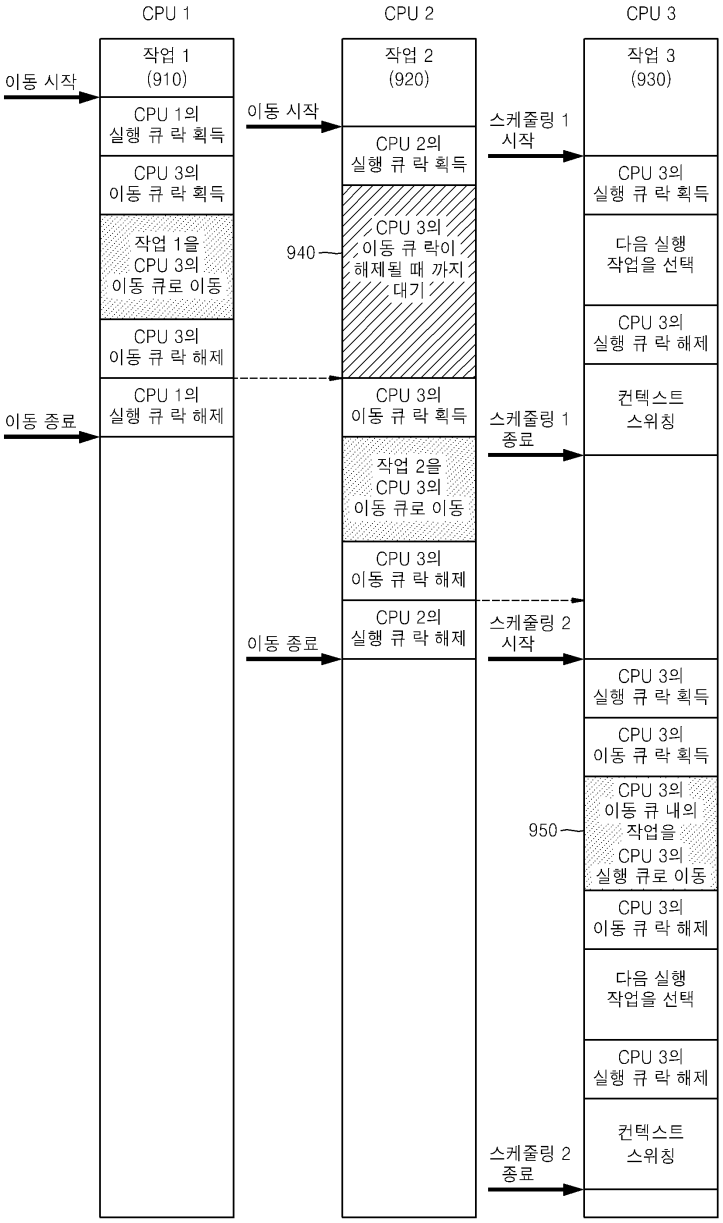
도면7


시작
701
로드 불균형 발생?
아니오
예
제1 CPU, 제2 CPU 및 이동시킬 작업을 선택
702
703
제1 CPU의 실행 큐에 대한 락을 획득?
아니오
대기
예
704
제2 CPU의 이동 큐에 대한 락을 획득?
아니오
대기
예
제1 CPU의 실행 큐로부터 제2 CPU의 이동 큐로 소정의 작업을 이동
705
제2 CPU의 이동 큐에 대한 락을 해제
706
제1 CPU의 실행 큐에 대한 락을 해제
707
종료

도면8


시작
801
제2 CPU의 실행 큐에 대한 락을 획득?
아니오
대기
예
802
제2 CPU의 이동 큐에 대한 락을 획득?
809
아니오
대기
예
803
제2 CPU의 이동 큐에 작업이 있는지?
아니오
예
제2 CPU의 이동 큐로부터 제2 CPU의 실행 큐로 소정의 작업을 이동
804
제2 CPU의 이동 큐에 대한 락을 해제
805
다음에 실행할 작업을 선택
806
제2 CPU의 실행 큐에 대한 락을 해제
807
컨텍스트 스위칭
808
종료

도면9


CPU 1
CPU 2
CPU 3
작업 1
(910)
작업 2
(920)
작업 3
(930)
이동 시작
CPU 1의
실행 큐 락 획득
이동 시작
CPU 2의
실행 큐 락 획득
스케줄링 1
시작
CPU 3의
실행 큐 락 획득
CPU 3의
이동 큐 락 획득
940
CPU 3의
이동 큐 락이
해제될 때 까지
대기
다음 실행
작업을 선택
작업 1을
CPU 3의
이동 큐로 이동
CPU 3의
실행 큐 락 해제
CPU 3의
이동 큐 락 해제
CPU 1의
실행 큐 락 해제
CPU 3의
이동 큐 락 획득
스케줄링 1
종료
컨텍스트
스위칭
이동 종료
작업 2을
CPU 3의
이동 큐로 이동
CPU 3의
이동 큐 락 해제
CPU 2의
실행 큐 락 해제
이동 종료
스케줄링 2
시작
CPU 3의
실행 큐 락 획득
CPU 3의
이동 큐 락 획득
950
CPU 3의
이동 큐 내의
작업을
CPU 3의
실행 큐로 이동
CPU 3의
이동 큐 락 해제
다음 실행
작업을 선택
CPU 3의
실행 큐 락 해제
컨텍스트
스위칭
스케줄링 2
종료