(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2012-0017294
(43) 공개일자 2012년02월28일

(51) Int. Cl.
*G06F 9/46* (2006.01) *G06F 15/17* (2006.01)
*G06F 9/44* (2006.01)
(21) 출원번호 10-2010-0079902
(22) 출원일자 2010년08월18일
심사청구일자 없음

(71) 출원인
**삼성전자주식회사**
경기도 수원시 영통구 삼성로 129 (매탄동)
(72) 발명자
**서성종**
경기도 화성시 동탄중앙로 213, 시범한빛마을금호어울림아파트 243동 1401호 (반송동)
**이승학**
경기도 용인시 기흥구 보정로 30, 동아솔레시티아파트 126동 1302호 (보정동)
*(뒷면에 계속)*
(74) 대리인
**특허법인 신지**

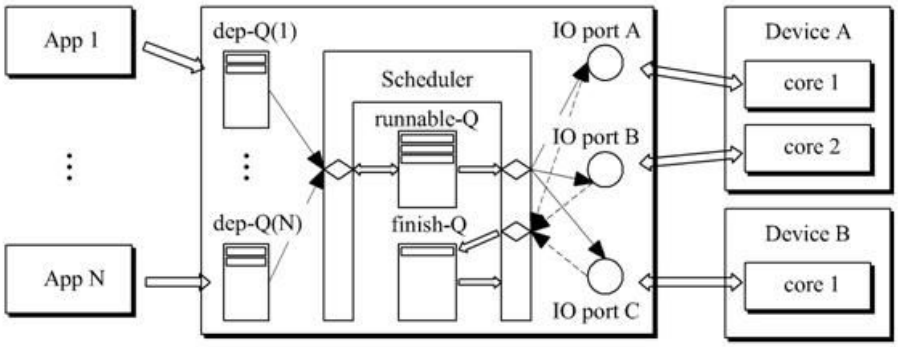
전체 청구항 수 : 총 20 항

# (54) 어플리케이션을 효율적으로 처리하는 스케쥴링 시스템 및 스케쥴링 방법

## *(57) 요 약*

여러 어플리케이션(application)이 공유하여 사용하는 멀티코어 시스템 상의 스케쥴링에 관한 기술이 개시된다. 본 발명의 일 양상에 따라, 코어가 동작하는 동안 의존성 제거(dependency resolving) 작업 및 워크(work)의 탐색 시간을 중첩시킬 수 있는 기술을 제공할 수 있다.

*대 표 도* - 도2



(72) 발명자

**임동우**

경기도 용인시 수지구 진산로 24, 송화마을 성원아파트 109동 406호 (상현동)

**송효정**

서울특별시 서대문구 모래내로 387-8 (홍제동)

**조승모**

서울특별시 서초구 양재동 한신휴플러스 902호

## *특허청구의 범위*

**청구항 1**

어플리케이션으로부터 워크를 전달받아 다수의 코어에서 실행되도록 할당하는 스케쥴링 시스템에 있어서,

상기 어플리케이션으로부터 전달받은 하나 이상의 워크를 상기 워크 간의 의존성(denpendency)을 고려하여 상기 코어에서 실행될 순서대로 저장하는 실행워크 저장부; 및

상기 코어에서 실행하여 완료된 워크를 저장하는 완료워크 저장부;를 포함하는 다수의 어플리케이션이 멀티코어를 공유하여 사용하도록 할당하는 스케쥴링 시스템.

**청구항 2**

제 1 항에 있어서,

상기 코어에서 실행하여 완료된 워크가 반환되는 경우 상기 완료된 워크를 상기 완료워크 저장부에 저장하고, 상기 어플리케이션으로부터 전달받은 워크가 상기 저장된 워크와 관련하여 의존성을 가진 경우 상기 의존성을 제거하여 상기 실행워크 저장부에 저장하는 스케쥴러;를 더 포함하는 스케쥴링 시스템.

**청구항 3**

제 1 항에 있어서,

하나 이상의 어플리케이션으로부터 전달받은 복수의 워크를 상기 실행워크 저장부에 저장하기 전에 상기 어플리케이션 별로 저장하는 임시저장부; 및

상기 실행워크 저장부에 저장된 워크를 상기 코어에 전달하거나 상기 코어에서 실행하여 완료된 워크를 상기 코어로부터 전달받는 입출력 포트;를 더 포함하는 스케쥴링 시스템.

**청구항 4**

제 3 항에 있어서,

각각의 임시저장부에 저장된 하나 이상의 워크 중에 의존성을 갖지 않은 워크들을 검색하여 상기 실행워크 저장부에 저장하고,

상기 실행워크 저장부에 저장된 워크들을 유휴상태의 코어에서 실행되도록 상기 입출력 포트를 통하여 전달하며,

상기 코어에서 실행되어 완료된 워크를 입출력 포트를 통하여 전달받아 상기 완료워크 저장부에 저장하는 스케쥴러;를 더 포함하는 스케쥴링 시스템.

**청구항 5**

제 4 항에 있어서, 상기 스케쥴러는

상기 임시저장부에 저장된 워크 중에서 상기 코어에서 실행되어 완료된 워크와 관련하여 의존성을 갖는 워크에 대하여 상기 의존성을 제거하는 스케쥴링 시스템.

**청구항 6**

제 4 항에 있어서, 상기 스케쥴러는

상기 코어에서 실행되어 완료된 워크를 전달받은 때에 실시간으로 상기 전달받은 워크와 관련하여 상기 임시저장부에 저장된 워크에 대한 의존성을 제거하거나, 일정 시간 단위로 상기 완료워크 저장부에 저장된 워크들과 관련하여 상기 임시저장부에 저장된 워크의 의존성을 제거하는 스케쥴링 시스템.

**청구항 7**

제 4 항에 있어서, 상기 스케쥴러는

상기 코어에서 실행되어 완료된 워크를 전달받은 때에, 상기 실행워크 저장부에 저장된 워크를 상기 완료된 워크를 전달한 코어에서 실행되도록 전달하는 스케쥴링 시스템.

**청구항 8**

제 5 항에 있어서, 상기 스케쥴러는

상기 워크들이 상기 코어에서 실행되고 있는 동안에, 상기 임시저장부에 저장된 워크 중에서 상기 코어에서 실행되어 완료된 워크와 관련하여 의존성을 갖는 워크에 대하여 상기 의존성을 제거하고, 상기 임시저장부에 저장된 워크 중에 의존성을 갖지 않은 워크들을 검색하여 상기 실행워크 저장부에 저장하는 스케쥴링 시스템.

**청구항 9**

어플리케이션으로부터 워크를 전달받아 다수의 코어에서 실행되도록 할당하는 스케쥴링 방법에 있어서,

상기 어플리케이션으로부터 전달받은 하나 이상의 워크를 상기 워크 간의 의존성(denpendency)을 고려하여 상기 코어에서 실행될 순서대로 저장하는 단계; 및

상기 코어에서 실행하여 완료된 워크를 저장하는 단계;를 포함하는 다수의 어플리케이션이 멀티코어를 공유하여 사용하도록 할당하는 스케쥴링 방법.

**청구항 10**

제 9 항에 있어서,

상기 코어에서 실행하여 완료된 워크가 반환되는 경우 상기 완료된 워크를 저장하고, 상기 어플리케이션으로부터 전달받은 워크가 상기 저장된 워크와 관련하여 의존성을 가진 경우 상기 의존성을 제거하는 단계;를 더 포함하는 스케쥴링 방법.

**청구항 11**

제 9 항에 있어서,

각각의 어플리케이션으로부터 전달된 하나 이상의 워크 중에 의존성을 갖지 않은 워크들을 검색하여 저장하는 단계;

상기 저장된 워크들을 유휴상태의 코어에서 실행되도록 상기 코어에 전달하는 단계; 및

상기 코어에서 실행하여 완료된 워크를 전달받아 저장하는 단계;를 더 포함하는 스케쥴링 방법.

**청구항 12**

제 11항에 있어서,

각각의 어플리케이션으로부터 전달된 하나 이상의 워크 중에서 상기 코어에서 실행되어 완료된 워크와 관련하여 의존성을 갖는 워크에 대하여 상기 의존성을 제거하는 단계;를 더 포함하는 스케쥴링 방법.

**청구항 13**

제 11 항에 있어서,

상기 코어에서 실행되어 완료된 워크를 전달받은 때에, 상기 저장된 워크를 상기 완료된 워크를 전달한 코어에서 실행되도록 전달하는 단계;를 더 포함하는 스케쥴링 방법.

**청구항 14**

제 11 항에 있어서,

상기 워크들이 상기 코어에서 실행되고 있는 동안에, 각각의 어플리케이션으로부터 전달된 하나 이상의 워크 중에서 상기 코어에서 실행되어 완료된 워크와 관련하여 의존성을 갖는 워크에 대하여 상기 의존성을 제거하고, 각각의 어플리케이션으로부터 전달된 하나 이상의 워크 중에 의존성을 갖지 않은 워크들을 검색하여 저장하는

단계;를 더 포함하는 스케쥴링 방법.

**청구항 15**

어플리케이션으로부터 전달받은 하나 이상의 워크를 상기 워크 간의 의존성(denpendency)을 고려하여 상기 코어에서 실행될 순서대로 저장하는 실행워크 저장부 및 상기 코어에서 실행하여 완료된 워크를 저장하는 완료워크 저장부를 포함하는 스케쥴링 시스템을 통한 다수의 어플리케이션이 멀티코어를 공유하여 사용하도록 할당하는 스케쥴링 방법에 있어서,

상기 실행워크 저장부에 저장된 워크가 존재하는지 판단하는 단계;

상기 실행워크 저장부에 저장된 워크가 존재한다면, 상기 워크를 실행할 수 있는 유휴상태의 코어가 존재하는지 판단하는 단계;

상기 유휴상태의 코어가 존재하는 경우, 상기 실행워크 저장부에 저장된 워크를 상기 유휴상태의 코어에서 실행되도록 전달하는 단계;

상기 실행워크 저장부에 저장된 워크가 존재하지 않거나 상기 유휴상태의 코어가 존재하지 않는 경우, 상기 완료워크 저장부에 완료된 워크가 존재하는지 판단하는 단계; 및

상기 완료워크 저장부에 완료된 워크가 존재한다면, 각각의 어플리케이션으로부터 전달된 하나 이상의 워크 중에서 상기 완료된 워크와 관련하여 의존성을 갖는 워크에 대하여 상기 의존성을 제거하는 단계;를 포함하는 스케쥴링 방법.

**청구항 16**

제 15 항에 있어서,

상기 완료워크 저장부에 완료된 워크가 존재하지 않으면, 각각의 어플리케이션으로부터 전달된 하나 이상의 워크 중에 의존성을 갖지 않는 워크들이 존재하는지 판단하는 단계; 및

상기 의존성을 갖지 않는 워크가 존재하는 경우, 상기 의존성을 갖지 않는 워크를 상기 실행워크 저장부에 저장하는 단계;를 포함하는 스케쥴링 방법

**청구항 17**

제 16 항에 있어서,

상기 유휴상태의 코어에서 실행되도록 전달하는 단계를 수행하고 난 후, 또는 상기 의존성을 갖지 않는 워크가 존재하지 않는 경우, 상기 실행워크 저장부에 저장된 워크가 존재하는지 판단하는 단계로 회기하는 스케쥴링 방법.

**청구항 18**

제 16 항에 있어서,

상기 의존성을 갖지 않는 워크가 존재하는 경우, 상기 워크를 전달한 어플리케이션으로부터 전달된 모든 워크를 획득하는 단계;

상기 획득된 워크들에 대해 상기 완료워크 저장부에 저장된 워크와 관련하여 의존성을 제거하는 단계; 및

상기 의존성이 제거된 워크를 상기 실행워크 저장부에 저장하는 단계;를 포함하는 스케쥴링 방법.

**청구항 19**

제 16 항에 있어서,

상기 유휴상태의 코어에서 실행되도록 전달하는 단계를 통하여 상기 코어에서 상기 워크가 실행되는 동안에, 상기 실행워크 저장부에 저장된 워크가 존재하는지 판단하는 단계 또는 그 이후의 단계가 수행되는 스케쥴링 방법.

**청구항 20**

제 18 항에 있어서,

상기 의존성을 갖지 않는 워크가 존재하는 경우, 상기 워크를 전달한 어플리케이션이 하나 이상 존재한다면 각각의 어플리케이션에 대한 모든 워크들에 대하여 상기 의존성을 제거하는 단계 및 상기 실행워크 저장부에 저장하는 단계를 반복하여 수행하는 스케쥴링 방법.

# 명 세 서

## 기 술 분 야

[0001] 본 발명은 여러 어플리케이션(application)이 공유하여 사용하는 멀티코어 시스템 내에, 의존성(denpendency) 기술자를 갖는 워크(work)가 대기하는 저장부(Queue 또는 Dependency queue)를 갖는 시스템 및 이러한 시스템의 운용방법에 관한 것으로, 보다 상세하게는 워크를 시스템에 요청할 수 있는 저장부(Dependency queue)를 어플리케이션에 제공하고, 어플리케이션으로 하여금 멀티코어 장치에서 수행되어야 할 워크를 저장부(Dependency queue)로부터 코어에 제공하도록 함으로서, 멀티코어 장치에서 워크가 실행될 때까지 시간을 단축하는 스케쥴링 시스템 및 방법에 관한 것이다.

## 배 경 기 술

[0002] 여러 어플리케이션(application)이 공유하여 사용하는 시스템을 스케쥴링 하는 기본적인 동작 과정은, 먼저 코어가 유휴상태에 있으면 완료된 워크를 기다리는 다른 모든 워크들과의 의존성을 제거하며, 이러한 과정을 의존성 제거(dependency resolving)이라 한다. 다음으로, 모든 저장부(dependency queue)를 탐색하면서 각각의 dependency queue에 대기 중인 워크를 모두 탐색한다. 이를 통하여 각각의 워크의 의존성이 모두 제거되었는지 확인한다. 만약 의존성이 제거된 워크가 존재한다면 유휴상태 코어로 워크의 수행을 요청한다.

[0003] 이렇게 완료된 워크에 대한 dependency resolving과정과 모든 dependency queue를 탐색하면서 의존성이 제거된 워크를 탐색하는 과정을 수행하는 시간 동안 코어는 계속 유휴상태로 있게 되어 시스템 전체의 효율성이 떨어지게 된다.

## 발명의 내용

### 해결하려는 과제

[0004] 시스템의 스케쥴링 동작에 따라서 외부 시스템과의 동작 효율성을 높이며, 코어가 유휴상태로 머물러있는 시간을 줄인다.

[0005] 대기 중인 워크 중 선행 워크가 모두 수행완료되면, 의존성이 모두 제거된 상태의 워크를 저장하는 저장부를 통하여 효율적인 멀티코어 시스템을 제공한다.

[0006] 수행완료된 선행 워크를 저장하는 저장부를 제공하여 의존성 제거작업과 의존성을 갖지 않는 워크를 탐색하는 작업을 수행하는 시간지연 없이 코어를 동작시킬 수 있다.

### 과제의 해결 수단

[0007] 본 발명의 일 양상에 따른 스케쥴링 시스템은 어플리케이션으로부터 워크를 전달받아 다수의 코어에서 실행되도록 할당하는 스케쥴링 시스템에 있어서, 어플리케이션으로부터 전달받은 하나 이상의 워크를 워크 간의 의존성(denpendency)을 고려하여 코어에서 실행될 순서대로 저장하는 실행워크 저장부 및 코어에서 실행하여 완료된 워크를 저장하는 완료워크 저장부를 포함할 수 있다.

[0008] 또한, 하나 이상의 어플리케이션으로부터 전달받은 복수의 워크를 실행워크 저장부에 저장하기 전에 어플리케이션 별로 저장하는 임시저장부 및 실행워크 저장부에 저장된 워크를 코어에 전달하거나 코어에서 실행하여 완료된 워크를 코어로부터 전달받는 입출력 포트를 포함할 수 있다.

[0009] 또한, 각각의 임시저장부에 저장된 하나 이상의 워크 중에 의존성을 갖지 않은 워크들을 검색하여 실행워크 저장부에 저장하고, 실행워크 저장부에 저장된 워크들을 유휴상태의 코어에서 실행되도록 입출력 포트를 통하여 전달하며, 코어에서 실행되어 완료된 워크를 입출력 포트를 통하여 전달받아 완료워크 저장부에 저장하는 스케쥴러를 포함할 수 있다.

[0010] 본 발명의 일 양상에 따른 스케쥴링 방법은 어플리케이션으로부터 워크를 전달받아 다수의 코어에서 실행되도록 할당하는 스케쥴링 방법에 있어서, 어플리케이션으로부터 전달받은 하나 이상의 워크를 워크 간의 의존성(denpendency)을 고려하여 코어에서 실행될 순서대로 저장하는 단계 및 코어에서 실행하여 완료된 워크를 저장하는 단계를 포함할 수 있다.

[0011] 본 발명의 다른 일 양상에 따른 스케쥴링 방법은 어플리케이션으로부터 전달받은 하나 이상의 워크를 상기 워크 간의 의존성(denpendency)을 고려하여 코어에서 실행될 순서대로 저장하는 실행워크 저장부 및 코어에서 실행하여 완료된 워크를 저장하는 완료워크 저장부를 포함하는 스케쥴링 시스템을 통한 다수의 어플리케이션이 멀티코어를 공유하여 사용하도록 할당하는 스케쥴링 방법에 있어서,

[0012] 실행워크 저장부에 저장된 워크가 존재하는지 판단하는 단계, 실행워크 저장부에 저장된 워크가 존재한다면, 워크를 실행할 수 있는 유휴상태의 코어가 존재하는지 판단하는 단계, 유휴상태의 코어가 존재하는 경우, 실행워크 저장부에 저장된 워크를 유휴상태의 코어에서 실행되도록 전달하는 단계, 실행워크 저장부에 저장된 워크가 존재하지 않거나 유휴상태의 코어가 존재하지 않는 경우, 완료워크 저장부에 완료된 워크가 존재하는지 판단하는 단계 및 완료워크 저장부에 완료된 워크가 존재한다면, 각각의 어플리케이션으로부터 전달된 하나 이상의 워크 중에서 완료된 워크와 관련하여 의존성을 갖는 워크에 대하여 의존성을 제거하는 단계를 포함할 수 있다.

## *발명의 효과*

[0013] 시스템의 스케쥴링 동작에 따라서 외부 시스템과의 동작 효율성을 향상하고, 코어가 유휴상태로 머물러있는 시간을 줄여 시스템 전체의 효율성을 높일 수 있다.

[0014] 의존성이 모두 제거된 상태의 워크를 저장하는 저장부를 제공하여 멀티코어 시스템의 성능을 향상할 수 있다.

[0015] 수행완료된 선행 워크를 저장하는 저장부를 제공하여 코어가 동작하고 있는 상태에서도 의존성 제거작업과 의존성을 갖지 않는 워크를 탐색하는 작업을 시간지연 없이 수행하여 멀티코어 시스템의 효율성을 높일 수 있다.

## *도면의 간단한 설명*

[0016] 도 1은 본 발명의 일 실시예에 따른 스케쥴링 시스템을 나타낸 블록도이다.

도 2는 본 발명의 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템을 나타낸 도면이다.

도 3은 본 발명의 일 실시예에 따른 스케쥴링 시스템의 임시저장부(dep-Q)에 저장된 워크 및 워크 간의 의존성을 나타낸 도면이다.

도 4는 본 발명의 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템에서 워크의 처리과정(Ⅰ)을 나타낸 도면이다.

도 5는 본 발명의 다른 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템에서 워크의 처리과정(Ⅱ)을 나타낸 도면이다.

도 6는 본 발명의 또 다른 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템에서 워크의 처리과정(Ⅲ)을 나타낸 도면이다.

도 7은 본 발명의 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템의 동작과정(Ⅰ)을 나타낸 흐름도이다.

도 8A 및 8B는 본 발명의 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템의 동작과정(Ⅱ)을 나타낸 흐름도이다.

도 9는 본 발명의 일 실시예에 따른 도 8B에 나타낸 (C)단계를 상세하게 나타낸 흐름도이다.

## *발명을 실시하기 위한 구체적인 내용*

[0017] 이하, 첨부된 도면을 참조하여 본 발명의 일 양상을 상세하게 설명한다.

[0018] 도 1은 본 발명의 일 실시예에 따른 스케쥴링 시스템을 나타낸 블록도이다. 도 1을 참조하면, 다수의 어플리케이션이 멀티코어를 공유하여 사용하도록 할당하는 스케쥴링 시스템은 스케쥴러(Scheduler, 100), 실행워크 저장부(runnable-Q, 110), 완료워크 저장부(finish-Q, 120), 임시저장부(dep-Q, 130), 입출력포트(IO-port, 140)를

포함할 수 있다.

[0019] 어플리케이션으로부터 워크를 전달받아 다수의 코어에서 실행되도록 할당하는 스케쥴링 시스템 간의 관계를 크게 두 가지로 나누면 첫 번째는 어플리케이션과 스케쥴링 시스템과의 동작 관계, 두 번째는 스케쥴링 시스템과 멀티코어 장치와의 동작 관계로 나눌 수 있다.

[0020] 전자의 동작관계는 어플리케이션들이 멀티코어 장치에서 병렬로 수행될 일을 스케쥴링 시스템의 저장부(dependency queue)에 입력(enqueue)하는 것으로, 후자의 동작관계는 스케쥴링 시스템이 어플리케이션들로부터 받은 워크(work) 중 의존성(dependency)을 갖지 않은 워크를 유휴상태 코어(idle core)에 실행 요청하고 워크의 실행이 완료된 코어는 스케쥴링 시스템에게 워크의 완료를 알리는 것으로 대표될 수 있다.

[0021] 스케쥴링 시스템은 멀티코어 장치와 별도의 코어를 구비한 장치에서 동작한다. 또한 어플리케이션과도 독립적인 실행 흐름을 갖는다. 즉, 저장부(dependency queue)에 워크를 입력(enqueue)하는 동작과, 멀티코어 장치에 요청된 워크의 실행 동작은 모두 스케쥴링 시스템과 병렬로 동작할 수 있다.

[0022] 실행워크 저장부(110)는 어플리케이션으로부터 전달받은 하나 이상의 워크를 워크 간의 의존성(denpendency)을 고려하여 코어에서 실행될 순서대로 저장한다.

[0023] 완료워크 저장부(120)는 코어에서 실행하여 완료된 워크를 저장한다.

[0024] 임시저장부(130)는 하나 이상의 어플리케이션으로부터 전달받은 복수의 워크를 실행워크 저장부(110)에 저장하기 전에 어플리케이션 별로 저장한다.

[0025] 입출력 포트(140)는 실행워크 저장부(110)에 저장된 워크를 코어에 전달하거나 코어에서 실행하여 완료된 워크를 코어로부터 전달받는다.

[0026] 스케쥴러(100)는 코어에서 실행하여 완료된 워크가 반환되는 경우 완료된 워크를 완료워크 저장부(120)에 저장하고, 어플리케이션으로부터 전달받은 워크가 저장된 워크와 관련하여 의존성을 가진 경우 의존성을 제거하여 실행워크 저장부(110)에 저장한다.

[0027] 스케쥴러(100)는 각각의 임시저장부(130)에 저장된 하나 이상의 워크 중에 의존성을 갖지 않은 워크들을 검색하여 실행워크 저장부(110)에 저장하고, 실행워크 저장부(110)에 저장된 워크들을 유휴상태의 코어에서 실행되도록 입출력 포트(140)를 통하여 전달하며, 코어에서 실행되어 완료된 워크를 입출력 포트(140)를 통하여 전달받아 완료워크 저장부(120)에 저장한다.

[0028] 또한, 스케쥴러(100)는 워크들이 코어에서 실행되고 있는 동안에, 임시저장부(130)에 저장된 워크 중에서 코어에서 실행되어 완료된 워크와 관련하여 의존성을 갖는 워크에 대하여 의존성을 제거하고, 임시저장부(130)에 저장된 워크 중에 의존성을 갖지 않은 워크들을 검색하여 실행워크 저장부(110)에 저장한다.

[0029] 스케쥴링 시스템은 이러한 장치를 통하여 여러 어플리케이션들로부터 입력된 워크들을 기술된 의존성(dependency) 순서에 맞게 동작시킬 수 있다.

[0030] 도 2는 본 발명의 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템을 나타낸 도면이다. 도 2를 참조하면, 본 발명의 일 양상이 적용된 스케쥴링 시스템은 어플리케이션이 전달하는 워크를 저장하는 임시저장부(dep-Q(1), dep-Q(N))와 워크를 코어에 입력하거나 코어로부터 출력하는 입출력포트(IO port) 사이에 스케쥴러(scheduler), 실행워크 저장부(runnable-Q) 및 완료워크 저장부(finish-Q)가 배치된다.

[0031] 실행워크 저장부(runnable-Q)는 의존성이 모두 제거된 상태의 워크들을 모아놓고, 완료워크 저장부(finish-Q)는 실행 완료된 워크를 모아놓는다.

[0032] 멀티코어 장치(Device A 또는 Device B)의 코어에서 실행하던 워크가 종료되면, 완료된 워크를 스케쥴링 시스템의 완료워크 저장부(finish-Q)에 저장하고(enqueue), 해당 코어(core)는 유휴상태(idle)로 전환된다. 그 즉시, 스케쥴러 시스템의 스케쥴러(scheduler)는 실행워크 저장부(runnable-Q)에 대기 중인 워크를 추출하여(dequeue) 바로 멀티코어 장치의 유휴상태 코어로 워크 실행을 요청한다.

[0033] 스케쥴러 시스템은 멀티코어 장치의 코어에서 워크가 실행됨과 동시에 완료워크 저장부(finish-Q)에 있는 모든 완료된 워크에 대해서 의존성 제거(dependency resolving) 작업을 수행한 후, 임시저장부(dep-Q)를 탐색한다. 각각의 임시저장부(dep-Q(1),…,dep-Q(N))에 대기 중인 워크를 모두 탐색하면서 의존성을 갖지 않은 워크들을 모두 실행워크 저장부(runnable-Q)로 옮긴다.

[0034] 이렇게 함으로써, 멀티코어 장치의 어떤 코어가 유휴상태가 되면, 실행워크 저장부(runnable-Q)에 대기 중인 워크를 의존성 제거(dependency resolving) 작업 또는 워크 탐색(searching runnable works) 과정에 의한 지연시간 없이 바로 멀티코어 장치의 유휴상태 코어에 워크의 실행을 요청할 수 있다.

[0035] 도 3은 본 발명의 일 실시예에 따른 스케쥴링 시스템의 임시저장부(dep-Q)에 저장된 워크 및 워크 간의 의존성을 나타낸 도면이다. 도 3을 참조하면 임시저장부(dep-Q)에 저장된 각각의 워크들은 의존성을 갖는 워크들로서 2개의 임시저장부(dep-Q(1), dep-Q(2))에 총 8개의 워크가 있음을 나타낸다.

[0036] 워크 간의 의존성과 관련하여 실행 가능 상태를 살펴보면, w1, w2, w5, w6은 어떠한 워크와도 의존성을 갖지 않으므로 바로 실행 가능한 상태이고, w3, w4는 w1이 실행 완료되어야 실행 가능한 상태(w1과 관련하여 의존성을 갖는 상태), w7, w8은 w5가 실행 완료되어야 실행 가능한 상태(w5와 관련하여 의존성을 갖는 상태)이다.

[0037] 도 3에 나타난 임시저장부(dep-Q1, dep-Q2)에 저장된 워크들 간의 관계를 바탕으로 이하의 도 4 내지 도 6의 멀티코어 시스템에서 워크의 처리과정을 설명한다.

[0038] 도 4는 본 발명의 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템에서 워크의 처리과정(Ⅰ)을 나타낸 도면이다.

[0039] 도 4의 일 양상의 멀티코어 시스템은 2개의 코어를 갖는 멀티코어 장치를 2개의 어플리케이션이 각각 dep-Q(dependency queue)를 이용하여 워크를 요청하는 과정을 나타낸다.

[0040] 어플리케이션은 dep-Q1에 w1, w2를, dep-Q2에 w5, w6을 입력(enqueue)한 다음 dep-Q1에 w3, w4를, dep-Q2에 w7, w8을 추가로 입력(enqueue)하고 있으며, device의 코어(core1, core2)의 초기상태는 유휴상태(idle)임을 나타낸다. 그리고 두 개의 dep-Q 모두 의존성(dependency)을 갖지 않은 워크를 가지고 있다면, 두 개의 dep-Q로부터 추출(dequeue)되는 기회는 동일하다고 가정한다.

[0041] 멀티코어 장치의 유휴상태 코어(idle core)에게 워크의 실행을 요청하기 위해서는 이미 실행 완료된 워크의 의존성 제거(dependency resolving) 작업과 모든 dep-Q들로부터 의존성을 갖지 않는 워크를 찾는 과정(searching runnable works)이 선행되어야 한다.

[0042] 스케쥴링 시스템은 멀티코어 장치의 core1이 idle임을 확인하고 모든 dep-Q들에 대해 searching runnable works 작업을 수행한다. 이러한 작업을 통하여 w1, w2, w5, w6이 각각의 dep-Q로부터 추출(dequeue)되고, runnable-Q에 입력(enqueue)된다. 스케쥴링 시스템은 runnable-Q로부터 w1을 추출(dequeue)하고 멀티코어 장치의 core1에 w1의 실행을 요청한다.

[0043] 요청이 끝남과 동시에 스케쥴링 시스템은 멀티코어 장치의 core2가 idle상태임을 확인하고 runnable-Q로부터 w5를 추출(dequeue)하고, core2에 w5의 실행을 요청한다. 멀티코어 장치의 두 코어가 동작 중인 상태에서 스케쥴링 시스템은 멀티코어 장치의 코어가 모두 idle상태가 아님을 확인하고 finish-Q와 dep-Q를 차례대로 탐색하면서 dependency resolving 작업과 searching runnable works 작업을 계속 시도한다.

[0044] 멀티코어 장치의 core1이 w1의 실행을 완료하면 finish-Q에 완료된 w1을 전달(enqueue)하고 idle상태로 전환된다. 그 즉시 스케쥴링 시스템은 runnable-Q로부터 w2를 추출(dequeue)하고 멀티코어 장치의 core1에 w2의 실행을 요청한다. 그와 동시에 멀티코어 장치의 core2가 w5의 실행을 완료하면 finish-Q에 완료된 w5를 전달(enqueue)하고 idle상태로 전환된다. 스케쥴링 시스템은 멀티코어 장치의 core2가 idle상태임을 확인하고 runnable-Q에서 w6을 추출(dequeue)하고 멀티코어 장치의 core2에 w6의 실행을 요청한다.

[0045] 멀티코어 장치의 두 core가 모두 동작 중인 상태에서 스케쥴링 시스템은 w1과 w5에 관련해서 dependency resolving 작업과 searching runnable works 작업을 연속적으로 수행한다. 이 과정에서 w3, w4, w7, w8이 모두 각각의 dep-Q로부터 runnable-Q로 옮겨진다.

[0046] 이런 과정을 반복함으로써 멀티코어 장치에서의 워크의 수행시간과 dependency resolving 작업 및 search runnable works 작업의 수행시간을 중첩시킬 수 있다. 또한 finish-Q를 두어 완료된 워크에 관련된 dependency resolving 작업보다 멀티코어 장치에서의 워크실행 요청을 먼저함으로써 스케쥴링 시스템과 외부 시스템과의 병렬성을 유지할 수 있다.

[0047] 도 5는 본 발명의 다른 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템에서 워크의 처리과정(Ⅱ)을 나타낸 도면이다.

[0048] 도 5의 일 양상의 멀티코어 시스템은 2개의 코어를 갖는 멀티코어 장치를 2개의 어플리케이션이 각각 dep-

Q(dependency queue)를 이용하여 워크를 요청하는 과정을 나타낸다.

[0049] 어플리케이션은 dep-Q1에 w1, w2를, dep-Q2에 w5, w6을 입력(enqueue)한 다음 dep-Q1에 w3, w4를, dep-Q2에 w7, w8을 추가로 입력(enqueue)하고 있으며, device의 코어(core1, core2)의 초기상태는 유휴상태(idle)임을 나타낸다. 그리고 두 개의 dep-Q 모두 의존성(dependency)을 갖지 않은 워크를 가지고 있다면, 두 개의 dep-Q로부터 추출(dequeue)되는 기회는 동일하다고 가정한다.

[0050] 도 5의 워크의 처리과정(Ⅱ)의 스케쥴링 시스템은 도 4의 처리과정(Ⅰ)과 동일하게, 멀티코어 장치의 core1이 idle임을 확인하고 모든 dep-Q들에 대해 searching runnable works 작업을 수행한다. 이러한 작업을 통하여 w1, w2, w5, w6이 각각의 dep-Q로부터 추출(dequeue)되고, runnable-Q에 입력(enqueue)된다. 스케쥴링 시스템은 runnable-Q로부터 w1을 추출(dequeue)하고 멀티코어 장치의 core1에 w1의 실행을 요청한다.

[0051] 요청이 끝남과 동시에 스케쥴링 시스템은 멀티코어 장치의 core2가 idle상태임을 확인하고 runnable-Q로부터 w5를 추출(dequeue)한다. core2에 w5의 실행을 요청한다. 멀티코어 장치의 두 코어가 동작 중인 상태에서 스케쥴링 시스템은 멀티코어 장치의 코어가 모두 idle상태가 아님을 확인하고 finish-Q와 dep-Q를 차례대로 탐색하면서 dependency resolving 작업과 searching runnable works 작업을 계속 시도한다.

[0052] 멀티코어 장치의 core1이 w1의 실행을 완료하면 finish-Q에 완료된 w1을 전달(enqueue)하고 idle상태로 전환된다. 그 즉시 스케쥴링 시스템은 runnable-Q로부터 w2를 추출(dequeue)하고 멀티코어 장치의 core1에 w2의 실행을 요청한다.

[0053] 멀티코어 장치의 core1은 w2가 실행되며, core2에서는 w5가 계속적으로 실행되고 있다. core2가 w5의 실행을 완료하면 finish-Q에 완료된 w5를 전달(enqueue)하고 idle상태로 전환된다. 스케쥴링 시스템은 멀티코어 장치의 core2가 idle상태임을 확인하고 runnable-Q에서 w6을 추출(dequeue)하고 멀티코어 장치의 core2에 w6의 실행을 요청한다.

[0054] 멀티코어 장치의 두 core가 모두 동작중인 상태에서 스케쥴링 시스템은 먼저 완료된 w1과 관련하여 dependency resolving 작업을 수행하던 중에, 완료된 w5가 전달되어 finish-Q에 저장되면, 수행하던 작업을 중단하고 idle core2에 w6을 실행할 것을 요청하여 다시 두 core가 모두 동작중인 상태에서, 완료된 w1과 w5에 관련해서 dependency resolving 작업과 searching runnable works 작업을 연속적으로 수행한다. 이 과정에서 w3, w4, w7, w8이 모두 각각의 dep-Q로부터 runnable-Q로 옮겨진다.

[0055] 이런 과정을 반복함으로써 멀티코어 장치에서의 워크의 수행시간과 dependency resolving 작업 및 search runnable works 작업의 수행시간을 중첩시킬 수 있다.

[0056] 도 6는 본 발명의 또 다른 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템에서 워크의 처리과정(Ⅲ)을 나타낸 도면이다.

[0057] 도 6의 일 양상의 멀티코어 시스템은 3개의 코어를 갖는 멀티코어 장치를 2개의 어플리케이션이 각각 dep-Q(dependency queue)를 이용하여 워크를 요청하는 과정을 나타낸다.

[0058] 어플리케이션은 dep-Q1에 w1, w2를, dep-Q2에 w5, w6을 입력(enqueue)한 다음 dep-Q1에 w3, w4를, dep-Q2에 w7, w8을 추가로 입력(enqueue)하고 있으며, device의 코어(core1, core2)의 초기상태는 유휴상태(idle)임을 나타낸다. 그리고 두 개의 dep-Q 모두 의존성(dependency)을 갖지 않은 워크를 가지고 있다면, 두 개의 dep-Q로부터 추출(dequeue)되는 기회는 동일하다고 가정한다.

[0059] 도 6의 워크의 처리과정(Ⅲ)의 스케쥴링 시스템은 도 4의 처리과정(Ⅰ)과 동일하게, 멀티코어 장치의 core1이 idle임을 확인하고 모든 dep-Q들에 대해 searching runnable works 작업을 수행한다. 이러한 작업을 통하여 w1, w2, w5, w6이 각각의 dep-Q로부터 추출(dequeue)되고, runnable-Q에 입력(enqueue)된다. 스케쥴링 시스템은 runnable-Q로부터 w1을 추출(dequeue)하고 멀티코어 장치의 core1에 w1의 실행을 요청한다.

[0060] 요청이 끝남과 동시에 스케쥴링 시스템은 멀티코어 장치의 core2가 idle상태임을 확인하고 runnable-Q로부터 w5를 추출(dequeue)한다. core2에 w5의 실행을 요청한다. 멀티코어 장치의 core3은 이전부터 w0를 실행하고 있는 상태이므로 멀티코어 장치의 세 코어가 동작 중인 상태가 되므로, 스케쥴링 시스템은 멀티코어 장치의 코어가 모두 idle상태가 아님을 인지하게 되고, finish-Q와 dep-Q를 차례대로 탐색하면서 dependency resolving 작업과 searching runnable works 작업을 계속 시도한다.

[0061] 멀티코어 장치의 core1이 w1의 실행을 완료하면 finish-Q에 완료된 w1을 전달(enqueue)하고 idle상태로 전환된

다. 그 즉시 스케쥴링 시스템은 runnable-Q로부터 w2를 추출(dequeue)하고 멀티코어 장치의 core1에 w2의 실행을 요청한다.

[0062] 그와 동시에 멀티코어 장치의 core3가 w0의 수행을 완료하면 finish-Q에 완료된 w0를 전달(enqueue)하고 idle상태로 전환된다. 스케쥴링 시스템은 멀티코어 장치의 core3가 idle상태임을 확인하고 runnable-Q에서 w6을 추출(dequeue)하고 멀티코어 장치의 core3에 w6의 실행을 요청한다. 따라서, 멀티코어 장치의 core1은 w2가 실행되며, core2에서는 w5가 계속적으로 수행되고, core3은 w6이 실행된다.

[0063] 멀티코어 장치의 세 core가 모두 동작 중인 상태에서 스케쥴링 시스템은 w0과 w1에 관련해서 dependency resolving 작업과 searching runnable works 작업을 연속적으로 수행한다. 이 과정에서 w3, w4, w7, w8이 모두 각각의 dep-Q로부터 runnable-Q로 옮겨진다.

[0064] 이러한 과정을 통해 멀티코어 장치의 세 core로부터 완료된 w2, w5, w6이 finish-Q에 저장(enqueue)되면 스케쥴링 시스템은 runnable-Q에 저장된 w3, w4, w7을 추출(dequeue)하여 세 core에 각각 전달한다. 다시 모든 core는 동작중인 상태가 되고, 스케쥴링 시스템은 w2, w5 및 w6과 관련해서 dependency resolving 작업과 searching runnable works 작업을 연속적으로 수행한다.

[0065] 이런 과정을 반복함으로써 다수의 코어 장치에서의 워크의 수행시간과 dependency resolving 작업 및 search runnable works 작업의 수행시간을 중첩시킬 수 있다.

[0066] 도 7은 본 발명의 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템의 동작과정(Ⅰ)을 나타낸 흐름도이다. 도 7을 참조하면, 멀티코어 시스템의 동작과정은 기본적으로 6단계로 이루어진다.

[0067] 먼저, 어플리케이션으로부터 전달받은 하나 이상의 워크를 워크 간의 의존성(denpendency)을 고려하여 코어에서 실행될 순서대로 저장한다(700). 저장된 차례대로 워크를 시스템 상의 유휴상태의 코어에 전달하여 실행요청하고(710), 전달된 워크는 코어에서 실행된다.

[0068] 코어에서 실행이 완료된 워크를 저장하고(720), 코어에서 실행될 순서대로 저장된 다음 워크(700단계에서 저장된 워크)를 상기 720단계에서 저장된 워크를 실행 완료하여 유휴상태가 된 코어에 전달하여 실행요청한다(730).

[0069] 상기 720단계에서 실행이 완료되어 저장된 워크와 관련하여 어플리케이션으로부터 전달받은 워크 중 의존성을 가진 워크의 의존성 제거(dependency resolving) 작업을 수행하고(740), 어플리케이션으로부터 전달받은 워크 중 의존성을 갖지 않은 워크들을 검색하여 코어에서 실행될 순서대로 저장한다(750).

[0070] 상기 750단계를 수행하면서 의존성을 갖지 않은 워크가 저장되는 경우 710단계부터 다시 수행하고, 저장된 워크가 없다면 어플리케이션으로부터 전달받은 새로운 워크가 없는 한 실행하여야 하는 워크가 없는 것이므로 동작을 종료하게 된다.

[0071] 도 8A 및 8B는 본 발명의 일 실시예에 따른 스케쥴링 시스템이 적용된 멀티코어 시스템의 동작과정(Ⅱ)을 나타낸 흐름도이다.

[0072] 전체적인 동작을 보면, 유휴상태 코어(idle core)에 신속하게 워크를 할당하기 위해 runnable-Q를 먼저 탐색하고, 완료된 work와 관련된 의존성 제거(dependency resolving) 작업을 한 후 dependency Q를 탐색하여 dependency가 모두 제거된 work를 runnable-Q로 이동시킨다. 이러한 과정을 반복 수행하므로서, 멀티코어 시스템을 스케쥴링하게 된다.

[0073] 보다 상세하게 동작과정을 살펴보면, 먼저, 시스템이 시작되면 runnable-Q를 탐색한다(801). 만약 runnable-Q에 저장된 work가 있으면 멀티코어 장치에 idle core가 있는지 확인한다(802). 만약 idle core가 있으면 runnable-Q에서 work를 dequeue한 후(803) 해당 core에 work의 실행을 요청하고(804) 스케쥴링 시스템은 runnable-Q가 비어있는지 다시 확인한다(801).

[0074] 만약 멀티코어 장치에 idle core가 없거나 runnable-Q가 비어있으면 스케쥴링 시스템은 완료된 work를 확인하기 위해 finish-Q를 탐색한다(805). finish-Q에 완료된 work가 있는 경우 이 work를 finish-Q에서 dequeue하고(806) 해당 work에 대한 dependency resolving 작업을 한다(807).

[0075] dependency resolving 작업이 완료되면 다시 finish-Q에 완료된 work가 있는지 확인한다(805). finish-Q가 비어있다면 스케쥴링 시스템은 dependency Q들을 탐색하면서 dependency를 갖지 않는 work를 찾는다(808). 만약 어떤 dependency Q에 dependency를 갖지 않는 work가 있다면, 해당 work를 dependency Q에서 dequeue하고(810) runnable-Q에 enqueue한다(811).

[0076] 스케쥴링 시스템은 이 과정을 반복함으로써(812), 여러 어플리케이션들로부터 입력된 work들을 기술된 dependency 순서에 맞게 동작시킬 수 있다. 또한 기존에 dependency Q만을 가지고 동작하는 시스템과는 달리, runnable-Q와 finish-Q를 두어 스케쥴링 시스템과 외부 시스템들과의 동기화 오버헤드(overhead)를 줄임과 동시에 dependency resolving 작업과 dependency Q들로부터 dependency를 갖지 않는 work를 탐색하는 작업을 외부 시스템과 병렬로 수행할 수 있다.

[0077] 도 9는 본 발명의 일 실시예에 따른 도 8B에 나타낸 (C)단계를 상세하게 나타낸 흐름도이다.

[0078] 도 8의 808단계에서 dependency Q에 dependency를 갖지 않은 work가 있는 경우, 스케쥴링 시스템 내의 어느 하나의 dependency Q(첫 번째 dependency Q)를 획득하고(900), 획득된 dependency Q에 저장된 work를 획득한다(901). 획득된 work에 대해 모두 dependency resolving 작업이 수행되었는지 확인하여(902), 모든 작업이 수행되었다면 dependency Q에서 의존성을 갖지 않은 work를 dequeue하여(903) runnable-Q에 work를 enqueue한다(904).

[0079] 첫 번째 dependency Q의 모든 work에 대해 runnable-Q로의 enqueue가 수행되도록 작업을 수행하기 위해서, runnable-Q에 enqueue된 work가 첫 번째 dependency Q에 존재하는 마지막 work인지 확인하여(905), 마지막 work가 아닌 경우 첫 번째 dependency Q에 저장된 다음 work를 획득하고(906) 획득한 work에 대하여 상기 902 단계부터 다시 수행한다.

[0080] 만약 마지막 work인 경우 상기 첫 번째 dependency Q가 스케쥴링 시스템 내에 존재하는 마지막 dependency Q인지 판단하여(907) 마지막 dependency Q가 아닌 경우에는 시스템 내에 존재하는 다음 dependenct Q를 획득하여(908) 획득한 dependency Q에 대하여 상기 901단계부터 다시 수행하고, 마지막 dependency Q인 경우에는 시스템의 전체 수행과정을 다시 수행할 것인지를 판단하여(812) 도 8A에 나타난 801단계부터 반복하여 수행하게 된다.

[0081] 도 8A 및 도 8B 또는 도 9에 나타난 일련의 처리과정에서, 804단계 또는 멀티코어 시스템 상의 상태에 의해 코어에서 work가 실행되고 있는 동안에 상기 805단계 내지 812단계 또는 상기 900단계 내지 907단계가 수행되므로써, 이전 work가 코어에서 실행되는 것과 동시에 시스템의 dependency resolving 작업 및 dependency를 갖지 않은 work를 탐색하는 작업을 수행할 수 있게 되어 스케쥴링 시스템의 지연 시간을 최소화 할 수 있다.

[0082] 한편, 본 발명의 실시 예들은 컴퓨터로 읽을 수 있는 기록 매체에 컴퓨터가 읽을 수 있는 코드로 구현하는 것이 가능하다. 컴퓨터가 읽을 수 있는 기록 매체는 컴퓨터 시스템에 의하여 읽혀질 수 있는 데이터가 저장되는 모든 종류의 기록 장치를 포함한다.

[0083] 컴퓨터가 읽을 수 있는 기록 매체의 예로는 ROM, RAM, CD-ROM, 자기 테이프, 플로피디스크, 광 데이터 저장장치 등이 있으며, 또한 캐리어 웨이브(예를 들어 인터넷을 통한 전송)의 형태로 구현하는 것을 포함한다. 또한, 컴퓨터가 읽을 수 있는 기록 매체는 네트워크로 연결된 컴퓨터 시스템에 분산되어, 분산 방식으로 컴퓨터가 읽을 수 있는 코드가 저장되고 실행될 수 있다. 그리고 본 발명을 구현하기 위한 기능적인(functional) 프로그램, 코드 및 코드 세그먼트들은 본 발명이 속하는 기술 분야의 프로그래머들에 의하여 용이하게 추론될 수 있다.

[0084] 이상에서 본 발명의 실시를 위한 구체적인 예를 살펴보았다. 전술한 실시 예들은 본 발명을 예시적으로 설명하기 위한 것으로 본 발명의 권리범위가 특정 실시 예에 한정되지 아니할 것이다.

## *부호의 설명*

[0085] 100 스케쥴러(Scheduler)

110 실행워크 저장부(runnable-Q)

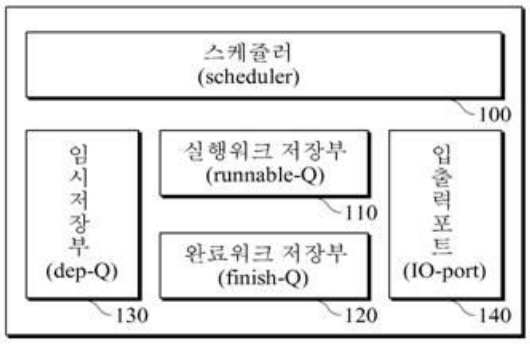
120 완료워크 저장부(finish-Q)

130 임시저장부(dep-Q)

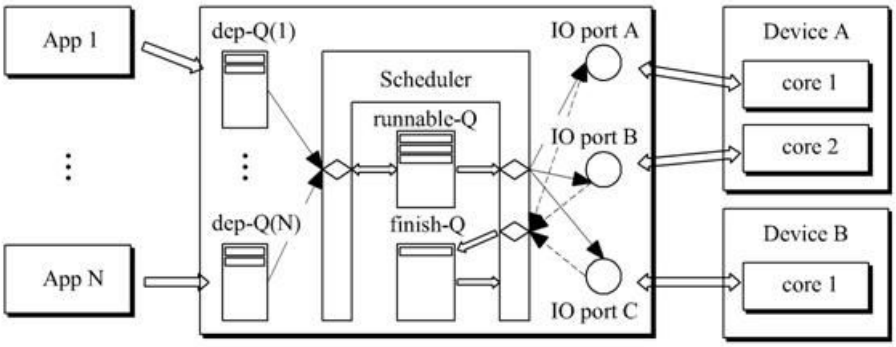
140 입출력포트(IO-port)

## 도면

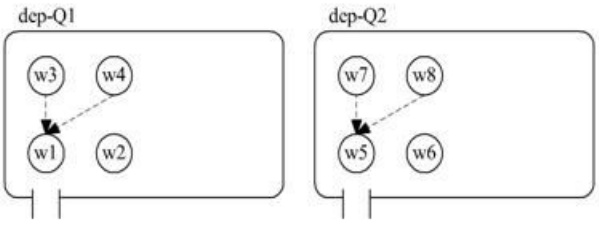
### 도면1



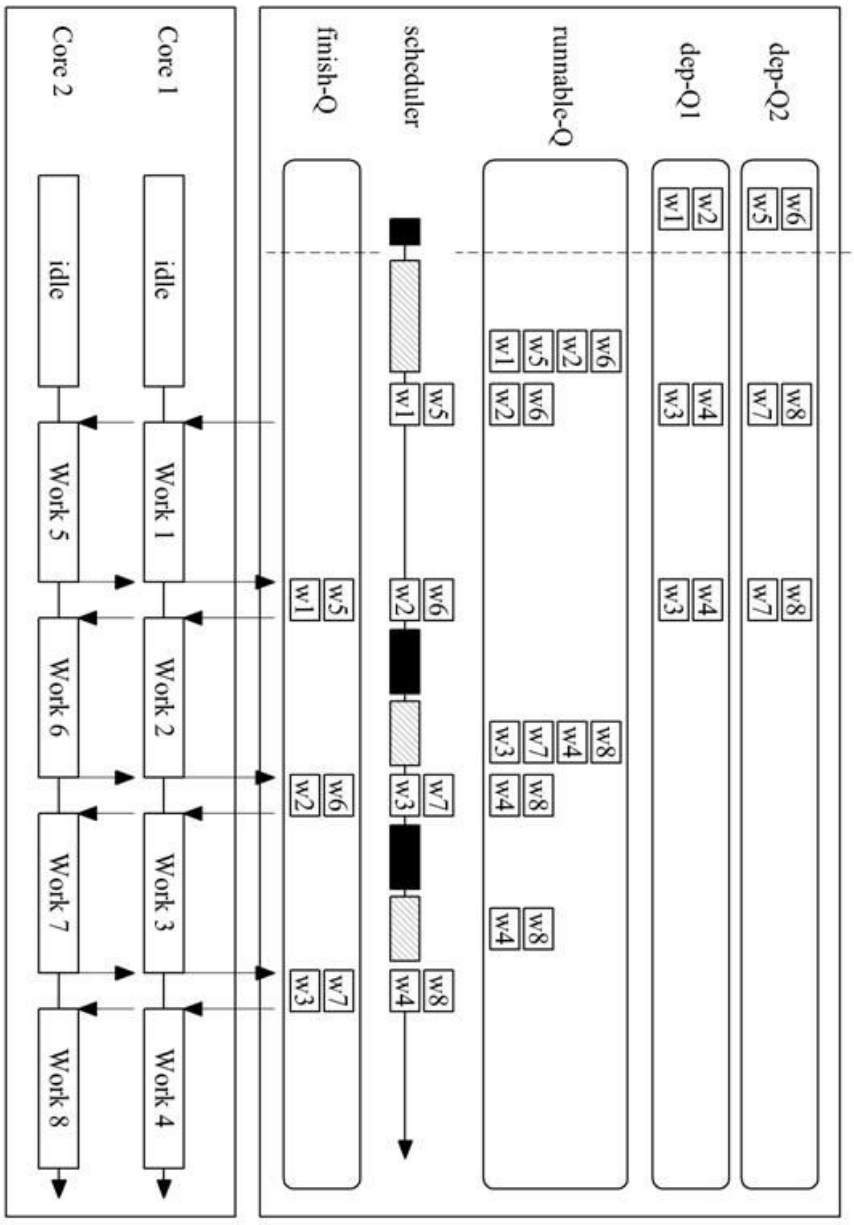

### 도면2



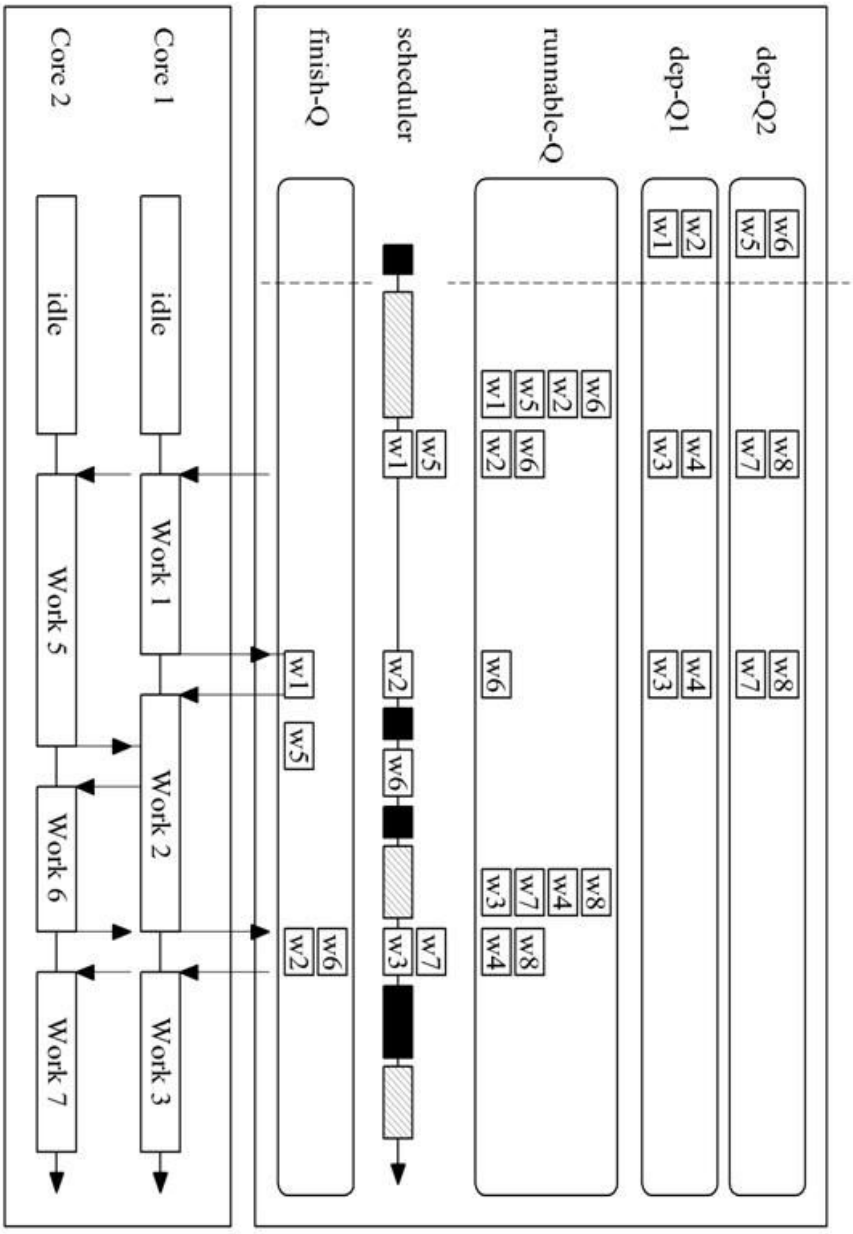

### 도면3



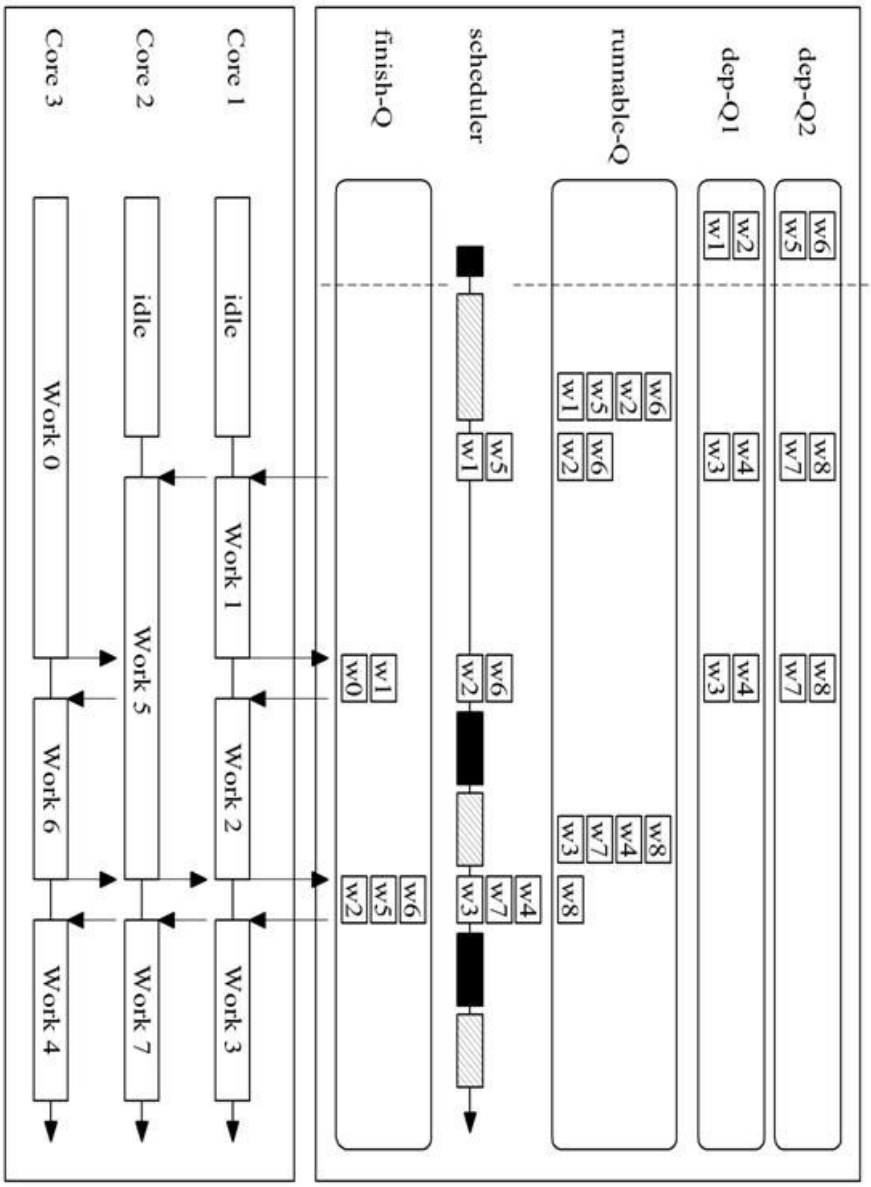
도면4


dep-Q2
dep-Q1
runnable-Q
scheduler
finish-Q
Core 1
Core 2
idle
idle
Work 1
Work 2
Work 3
Work 4
Work 5
Work 6
Work 7
Work 8

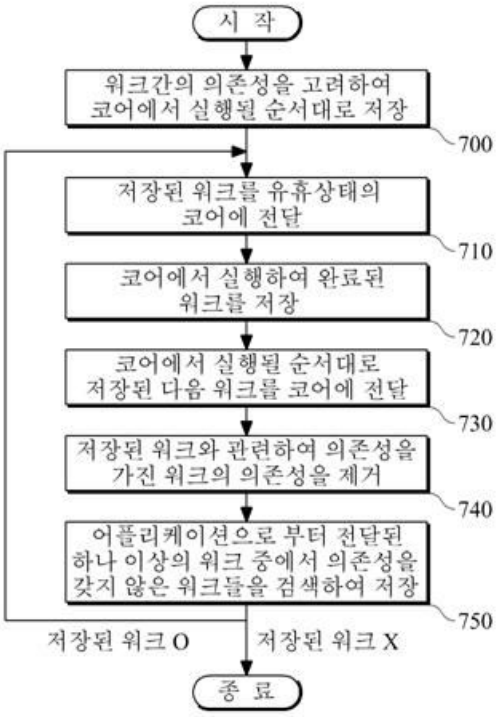
도면5


dep-Q2
dep-Q1
runnable-Q
scheduler
finish-Q
Core 1
Core 2
idle
idle
Work 1
Work 5
Work 2
Work 6
Work 3
Work 7

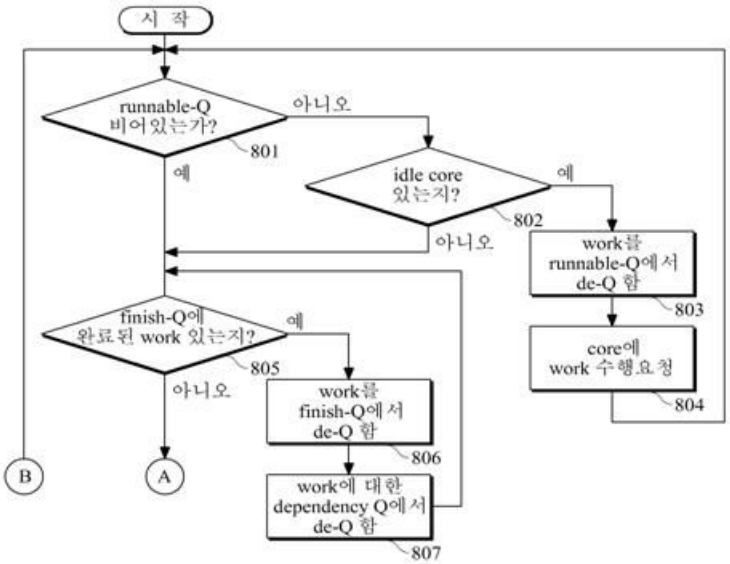
도면6


dep-Q2
dep-Q1
runnable-Q
scheduler
finish-Q
Core 1
Core 2
Core 3
idle
idle
Work 0
Work 1
Work 5
Work 2
Work 6
Work 3
Work 7
Work 4

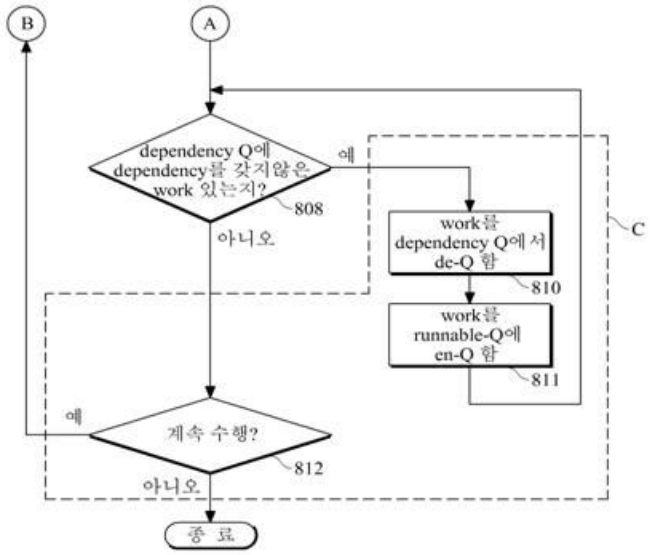
도면7


시 작
워크간의 의존성을 고려하여 코어에서 실행될 순서대로 저장
700
저장된 워크를 유휴상태의 코어에 전달
710
코어에서 실행하여 완료된 워크를 저장
720
코어에서 실행될 순서대로 저장된 다음 워크를 코어에 전달
730
저장된 워크와 관련하여 의존성을 가진 워크의 의존성을 제거
740
어플리케이션으로 부터 전달된 하나 이상의 워크 중에서 의존성을 갖지 않은 워크들을 검색하여 저장
750
저장된 워크 O
저장된 워크 X
종 료

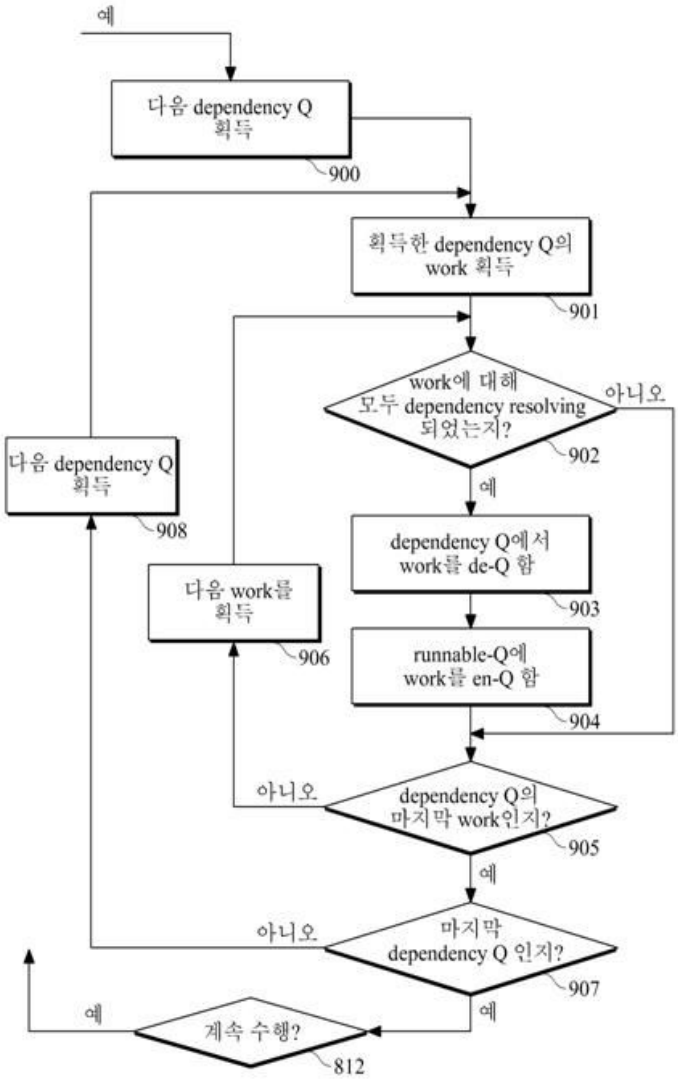
도면8a


시 작
runnable-Q 비어있는가?
801
아니오
예
idle core 있는지?
802
예
아니오
work를 runnable-Q에서 de-Q 함
803
core에 work 수행요청
804
finish-Q에 완료된 work 있는지?
805
예
아니오
work를 finish-Q에서 de-Q 함
806
work에 대한 dependency Q에서 de-Q 함
807
B
A

도면8b


B
A
dependency Q에 dependency를 갖지않은 work 있는지?
808
예
아니오
work를 dependency Q에서 de-Q 함
810
work를 runnable-Q에 en-Q 함
811
C
예
계속 수행?
812
아니오
종 료

도면9


예
다음 dependency Q 획득
900
획득한 dependency Q의 work 획득
901
work에 대해 모두 dependency resolving 되었는지?
902
아니오
예
dependency Q에서 work를 de-Q 함
903
runnable-Q에 work를 en-Q 함
904
dependency Q의 마지막 work인지?
905
아니오
예
마지막 dependency Q 인지?
907
아니오
예
계속 수행?
812
예
다음 dependency Q 획득
908
다음 work를 획득
906