



(19) 대한민국특허청(KR)

(12) 등록특허공보(B1)

(45) 공고일자 2025년03월04일

(11) 등록번호 10-2776397

(24) 등록일자 2025년02월27일

(51) 국제특허분류(Int. Cl.)
G06F 9/50 (2018.01) G06F 15/78 (2006.01)
G06F 9/4401 (2018.01)

(52) CPC특허분류
G06F 9/5077 (2013.01)
G06F 15/78 (2013.01)

(21) 출원번호 10-2023-7045014

(22) 출원일자(국제) 2023년04월28일
심사청구일자 2023년12월27일

(85) 번역문제출일자 2023년12월27일

(65) 공개번호 10-2024-0159790

(43) 공개일자 2024년11월06일

(86) 국제출원번호 PCT/CN2023/091876

(87) 국제공개번호 WO 2024/221468

국제공개일자 2024년10월31일

(56) 선행기술조사문헌

한국공개특허 제10-2013-0104958호(2013.09.25.)
1부.*

(뒷면에 계속)

전체 청구항 수 : 총 18 항

심사관 : 김중기

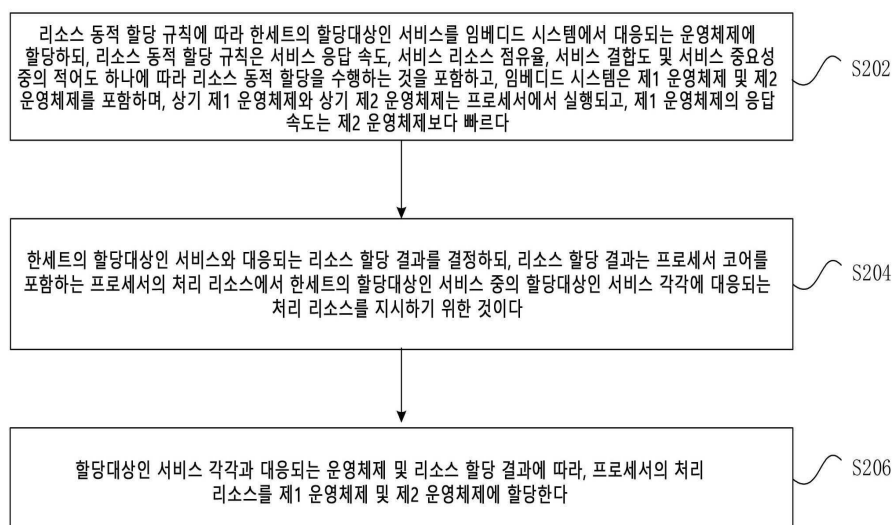
(54) 발명의 명칭 임베디드 시스템의 실행 방법 및 장치, 임베디드 시스템 및 칩

(57) 요약

본 출원은 임베디드 시스템의 실행 방법 및 장치, 임베디드 시스템 및 칩을 개시한다. 상기 방법은, 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당하되, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 상기 제1 운영체제와 상기 제2 운영체제는 프로세서에서 실행되고, 제1 운영체제의 응답 속도는 제2 운영체제보다 빠르다

(뒷면에 계속)

대표도 - 도2



에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 상기 제1 운영체제의 응답 속도는 상기 제2 운영체제보다 빠른 것인 단계; 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하되, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것인 단계; 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 단계; 를 포함한다.

(52) CPC특허분류

G06F 9/441 (2013.01)

G06F 9/5027 (2013.01)

G06F 9/5083 (2013.01)

(72) 발명자

리우, 바오양

중국 지양수 215000 쑤저우 우중 존 귀샹 스트리트
관푸 로드 넘버 1 빌딩 9

첸, 차오판

중국 지양수 215000 쑤저우 우중 존 귀샹 스트리트
관푸 로드 넘버 1 빌딩 9

마, 웬카이

중국 지양수 215000 쑤저우 우중 존 귀샹 스트리트
관푸 로드 넘버 1 빌딩 9

(56) 선행기술조사문헌

한국등록특허 제10-1557995호(2015.10.19.) 1부.*

한국등록특허 제10-1934293호(2019.01.02.) 1부.*

한국공개특허 제10-2017-0110615호(2017.10.11.)

1부.*

한국공개특허 제10-2014-0076185호(2014.06.20.)

1부.*

*는 심사관에 의하여 인용된 문헌

명세서

청구범위

청구항 1

프로세서에서 실행되는 제1 운영체제 및 제2 운영체제;

리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 서비스 관리 모듈;

상기 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 리소스 동적 할당 모듈;

상기 할당 대상인 서비스 각각과 대응되는 운영체제 및 상기 리소스 할당 결과에 따라, 상기 프로세서의 처리 리소스를 상기 제1 운영체제 및 상기 제2 운영체제에 할당하는 리소스 적응형 스케줄링 모듈; 을 포함하되,

상기 제1 운영체제의 응답 속도는 상기 제2 운영체제보다 빠르고,

상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당을 수행하며,

상기 리소스 할당 결과는 상기 프로세서의 처리 리소스에서 상기 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 상기 프로세서의 처리 리소스는 프로세서 코어를 포함하며,

상기 서비스 관리 모듈은,

상기 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스를 상기 제1 운영체제에 할당하고, 상기 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스를 상기 제2 운영체제에 할당하는 단계;

상기 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비스를 상기 제1 운영체제에 할당하고, 상기 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당 대상인 서비스를 상기 제2 운영체제에 할당하는 단계;

상기 한세트의 할당 대상인 서비스에 민감한 정보가 포함되는 할당 대상인 서비스를 타깃 운영체제에 할당하되, 상기 타깃 운영체제는 상기 제1 운영체제 및 상기 제2 운영체제에서, 사용 객체와 인터랙션하는 빈도가 낮은 운영체제인 것인 단계; 중의 적어도 한 단계를 실행하여 상기 리소스 동적 할당 규칙에서 서비스 응답 속도와 대응되는 할당 규칙에 따라 상기 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

청구항 2

제1항에 있어서,

규칙 구성 파일을 읽고, 상기 리소스 동적 할당 규칙을 기록하기 위한 규칙 구조체를 생성하는 로드 밸런스 전략 모듈을 더 포함하는 것을 특징으로 하는 임베디드 시스템.

청구항 3

제2항에 있어서,

상기 로드 밸런스 전략 모듈은 또한 상기 제2 운영체제의 외부 인터페이스를 통해 규칙 업데이트 구성 파일을 획득하며,

상기 규칙 업데이트 구성 파일은 구성된 상기 리소스 동적 할당 규칙을 업데이트하기 위한 것이고, 상기 규칙

업데이트 구성 파일을 사용하여 상기 규칙 구조체를 업데이트하여, 상기 규칙 구조체에 기록된 상기 리소스 동적 할당 규칙을 업데이트하는 것을 특징으로 하는 임베디드 시스템.

청구항 4

제1항에 있어서,

상기 서비스 관리 모듈은 또한,

상기 한세트의 할당 대상인 서비스에서 상기 제1 운영체제의 할당된 서비스와의 서비스 결합도가 제1 결합도 임계값 이상인 할당 대상인 서비스를 상기 제1 운영체제에 할당하는 단계;

상기 한세트의 할당 대상인 서비스에서 상기 제2 운영체제의 할당된 서비스와의 서비스 결합도가 제2 결합도 임계값 이상인 할당 대상인 서비스를 상기 제2 운영체제에 할당하는 단계; 중의 적어도 한 단계를 실행하여 상기 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

청구항 5

제1항에 있어서,

상기 리소스 동적 할당 모듈은,

상기 서비스 관리 모듈의 할당 결과에 따라, 상기 제1 운영체제의 처리 리소스의 리소스 이용 상황 및 상기 제2 운영체제의 처리 리소스의 리소스 이용 상황을 결합하여, 상기 한세트의 할당 대상인 서비스와 상기 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성하는 단계를 실행하여 상기 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

청구항 6

제1항에 있어서,

상기 리소스 동적 할당 모듈은,

프로세서 코어를 단위로 상기 프로세서의 처리 리소스를 상기 제1 운영체제 및 상기 제2 운영체제에 할당하는 것을 특징으로 하는 임베디드 시스템.

청구항 7

제1항에 있어서,

상기 리소스 적응형 스케줄링 모듈은,

상기 리소스 할당 결과에 따라 상기 프로세서의 처리 리소스에서의 할당되지 않은 처리 리소스에는 대응되는 할당 대상인 서비스가 존재하는 것으로 결정할 경우, 상기 할당되지 않은 처리 리소스를 상기 할당되지 않은 처리 리소스와 대응되는 할당 대상인 서비스에 의해 할당되는 운영체제에 할당하는 단계를 실행하여 상기 할당 대상인 서비스 각각과 대응되는 운영체제 및 상기 리소스 할당 결과에 따라, 상기 프로세서의 처리 리소스를 상기 제1 운영체제 및 상기 제2 운영체제에 할당하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

청구항 8

제1항에 있어서,

상기 제1 운영체제 및 상기 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하기 위한 리소스 선점과

해제 모듈을 더 포함하고,

상기 리소스 선점과 해제 모듈은,

코어 간 통신 인터페이스를 통해 상기 제1 운영체제 및 상기 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것을 특징으로 하는 임베디드 시스템.

청구항 9

제8항에 있어서,

상기 리소스 선점과 해제 모듈은,

코어 간 통신 인터페이스를 통해 상기 제1 운영체제의 제1 인터랙션 요청을 상기 제2 운영체제에 전송하되, 상기 제1 인터랙션 요청은 상기 제2 운영체제와 리소스 인터랙션을 수행하는 것을 요청하기 위한 것이고, 상기 리소스 인터랙션은 리소스 선점 및 리소스 해제 중의 하나를 포함하는 것인 단계;

상기 코어 간 통신 인터페이스를 통해 상기 제2 운영체제에서 상기 제1 인터랙션 요청에 응답하여 반환하는 제1 인터랙션 응답을 획득하되, 상기 제1 인터랙션 응답은 상기 제1 운영체제에서 상기 제1 인터랙션 응답에 따라 상기 제2 운영체제와 상기 리소스 인터랙션을 수행하는 것을 지시하기 위한 것인 단계; 를 실행하여 코어 간 통신 인터페이스를 통해 상기 제1 운영체제 및 상기 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

청구항 10

제9항에 있어서,

제1 시스템 제어 모듈을 더 포함하고,

제1 시스템 제어 모듈은,

상기 제1 운영체제의 처리 리소스의 리소스 이용율을 검출하고, 상기 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 상기 제1 운영체제에서 상기 제2 운영체제와 상기 리소스 인터랙션을 수행하는 것으로 결정할 경우, 상기 리소스 선점과 해제 모듈에서 상기 코어 간 통신 인터랙션을 통해 상기 제1 인터랙션 요청을 상기 제2 운영체제에 전송하는 것을 트리거하며,

또한, 상기 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 상기 제1 운영체제에서 상기 제2 운영체제의 처리 리소스를 선점하는 것으로 결정하는 단계;

상기 제1 운영체제의 처리 리소스가 유휴 상태이고, 상기 제1 운영체제에서 실행 대상인 서비스가 존재하지 않을 경우, 상기 제1 운영체제에서 상기 제2 운영체제로 처리 리소스를 해제하는 것을 대기하는 것으로 결정하는 단계; 중의 적어도 한 단계를 실행하는 것을 특징으로 하는 임베디드 시스템.

청구항 11

제9항에 있어서,

상기 리소스 선점과 해제 모듈은,

코어 간 통신 인터페이스를 통해 중단 번호가 제1 중단 번호인 제1 선점 요청을 상기 제2 운영체제에 전송하되, 상기 제1 선점 요청은 상기 제2 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것인 단계;

코어 간 통신 인터페이스를 통해 중단 번호가 제2 중단 번호인 리소스 해제 요청을 상기 제2 운영체제에 전송하되, 상기 리소스 해제 요청은 상기 제2 운영체제에 상기 제1 운영체제에 의해 점유된 처리 리소스를 해제하는 것을 요청하기 위한 것인 단계; 중의 적어도 한 단계를 실행하여 코어 간 통신 인터페이스를 통해 상기 제1 운영체제의 제1 인터랙션 요청을 상기 제2 운영체제에 전송하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

템.

청구항 12

제8항에 있어서,

상기 리소스 선점과 해제 모듈은,

코어 간 통신 인터페이스를 통해 상기 제2 운영체제의 제2 인터랙션 요청을 상기 제1 운영체제에 전송하되, 상기 제2 인터랙션 요청은 상기 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것인 단계;

상기 코어 간 통신 인터페이스를 통해 상기 제1 운영체제에서 상기 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하되, 상기 제2 인터랙션 응답은 상기 제1 운영체제가 상기 제2 운영체제에서 상기 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는지 여부를 지시하기 위한 것인 단계; 를 실행하여 코어 간 통신 인터페이스를 통해 상기 제1 운영체제와 상기 제2 운영체제 사이에서 처리 리소스의 서점과 해제를 수행하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

청구항 13

제12항에 있어서,

상기 제2 운영체제의 처리 리소스의 리소스 이용율을 검출하기 위한 제2 시스템 제어 모듈을 더 포함하되,

상기 제2 운영체제의 처리 리소스의 리소스 이용율에 따라, 상기 제2 운영체제에서 상기 제1 운영체제의 처리 리소스를 선점하는 것을 결정할 경우, 상기 리소스 선점과 해제 모듈에서 상기 코어 간 통신 인터페이스를 통해 상기 제2 인터랙션 요청을 상기 제1 운영체제에 전송하도록 트리거하는 것을 특징으로 하는 임베디드 시스템.

청구항 14

제13항에 있어서,

상기 제2 시스템 제어 모듈은 또한,

상기 제2 운영체제의 처리 리소스의 현재의 리소스 이용율이 제2 이용율 임계값 이상인 경우, 상기 제2 운영체제에서 상기 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정하는 단계;

상기 제2 운영체제의 실행 중 서비스 및 상기 제2 운영체제의 실행 대상인 서비스에 따라 상기 제2 운영체제의 처리 리소스의 리소스 이용율이 제3 이용율 임계값 이상인 경우, 상기 제2 운영체제에서 상기 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정하는 단계; 중의 적어도 한 단계를 실행하는 것을 특징으로 하는 임베디드 시스템.

청구항 15

제12항에 있어서,

상기 리소스 선점과 해제 모듈은,

코어 간 통신 인터페이스를 통해 중단 번호가 제3 중단 번호인 제2 선점 요청을 상기 제1 운영체제에 전송하되, 상기 제2 선점 요청은 상기 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것인 단계를 실행하여 코어 간 통신 인터페이스를 통해 상기 제2 운영체제의 제2 인터랙션 요청을 상기 제1 운영체제에 전송하는 것을 실현하거나, 또는

코어 간 통신 인터페이스를 통해 중단 번호가 제4 중단 번호인 리소스 해제를 허용하는 응답을 상기 제2 운영체제에 전송하되, 상기 리소스 해제를 허용하는 응답은 상기 제1 운영체제에서는 상기 제2 운영체제에서 상기 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것을 지시하기 위한 것인 단계;

코어 간 통신 인터페이스를 통해 중단 번호가 제5 중단 번호인 리소스 해제를 거부하는 응답을 상기 제2 운영체제에 전송하되, 상기 리소스 해제를 거부하는 응답은 상기 제1 운영체제에서는 상기 제2 운영체제에서 상기 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것을 지시하기 위한 것인 단계; 중의 적어도 한 단계를 실행하여 상기 코어 간 통신 인터페이스를 통해 상기 제1 운영체제에서 상기 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

청구항 16

칩 및 적어도 2개의 운영체제를 포함하며,

상기 칩은 프로세서, 하드웨어 제어기, 제1 버스 및 제2 버스를 포함하되,

상기 제1 버스의 대역폭은 상기 제2 버스의 대역폭보다 높고, 또한 상기 제1 버스는 멀티 마스터 멀티 슬레이브 모드로 구성되고, 상기 제2 버스는 단일 마스터 멀티 슬레이브 모드로 구성되며,

상기 적어도 2개의 운영체제는 상기 프로세서에 기반하여 실행되고, 상기 프로세서의 처리 리소스는 상기 적어도 2개의 운영체제에 동적 할당되고, 상기 프로세서의 처리 리소스는 프로세서 코어를 포함하며,

상기 적어도 2개의 운영체제는 상기 제1 버스를 통해 통신을 수행하며;

상기 적어도 2개의 운영체제는 상기 제2 버스를 통해 상기 하드웨어 제어기에 대한 제어를 실현하고,

상기 적어도 2개의 운영체제는, 상기 프로세서의 적어도 하나의 프로세서 코어에서 실행되는 제1 운영체제 및 적어도 하나의 프로세서 코어에서 실행되는 제2 운영체제를 포함하고, 상기 제1 운영체제의 응답 속도가 상기 제2 운영체제보다 빠르고,

상기 제2 운영체제는,

리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 서비스 관리 모듈;

상기 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 리소스 동적 할당 모듈;

상기 할당 대상인 서비스 각각과 대응되는 운영체제 및 상기 리소스 할당 결과에 따라, 상기 프로세서의 처리 리소스를 상기 제1 운영체제 및 상기 제2 운영체제에 할당하는 리소스 적응형 스케줄링 모듈; 을 포함하되,

상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당을 수행하고,

상기 리소스 할당 결과는 상기 프로세서의 처리 리소스에서 상기 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하며,

상기 서비스 관리 모듈은,

상기 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스를 상기 제1 운영체제에 할당하고, 상기 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스를 상기 제2 운영체제에 할당하는 단계;

상기 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비스를 상기 제1 운영체제에 할당하고, 상기 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당 대상인 서비스를 상기 제2 운영체제에 할당하는 단계;

상기 한세트의 할당 대상인 서비스에 민감한 정보가 포함되는 할당 대상인 서비스를 타깃 운영체제에 할당하되, 상기 타깃 운영체제는 상기 제1 운영체제 및 상기 제2 운영체제에서, 사용 객체와 인터랙션하는 빈도가 낮은 운영체제인 것인 단계; 중의 적어도 한 단계를 실행하여 상기 리소스 동적 할당 규칙에서 서비스 응답 속도와 대응되는 할당 규칙에 따라 상기 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현하는 것을 특징으로 하는 임베디드 시스템.

청구항 17

리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당 하되, 상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요 성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 상기 임베디드 시스템은 제1 운영체 제 및 제2 운영체제를 포함하며, 상기 제1 운영체제와 상기 제2 운영체제는 프로세서에서 실행되고, 상기 제1 운영체제의 응답 속도는 상기 제2 운영체제보다 빠른 것인 단계;

상기 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하되, 상기 리소스 할당 결과는 상기 프 로세서의 처리 리소스에서 상기 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 상기 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것인 단계;

상기 할당 대상인 서비스 각각과 대응되는 운영체제 및 상기 리소스 할당 결과에 따라, 상기 프로세서의 처리 리소스를 상기 제1 운영체제 및 상기 제2 운영체제에 할당하는 단계; 를 포함하고,

상기 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대 상인 서비스를 상기 제1 운영체제에 할당하고, 상기 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사 항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스를 상기 제2 운영체제에 할당하는 단계;

상기 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비 스를 상기 제1 운영체제에 할당하고, 상기 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유 율 임계값 이상인 할당 대상인 서비스를 상기 제2 운영체제에 할당하는 단계;

상기 한세트의 할당 대상인 서비스에 민감한 정보가 포함되는 할당 대상인 서비스를 타깃 운영체제에 할당하되, 상기 타깃 운영체제는 상기 제1 운영체제 및 상기 제2 운영체제에서, 사용 객체와 인터랙션하는 빈도가 낮은 운 영체제인 것인 단계; 중의 적어도 한 단계를 실행하여 상기 리소스 동적 할당 규칙에서 상기 서비스 응답 속도와 대응되는 할당 규칙에 따라 상기 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현하 는 것을 특징으로 하는 임베디드 시스템의 실행 방법.

청구항 18

리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당 하는 제1 할당 유닛;

상기 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 제1 결정 유닛;

상기 할당 대상인 서비스 각각과 대응되는 운영체제 및 상기 리소스 할당 결과에 따라, 프로세서의 처리 리소스 를 제1 운영체제 및 제2 운영체제에 할당하는 제2 할당 유닛; 을 포함하되,

상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 상기 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 상기 제1 운영체제와 상기 제2 운영체제는 프로세서에서 실행되고, 상기 제1 운영체제의 응답 속도는 상기 제2 운영체제보다 빠르고,

상기 리소스 할당 결과는 상기 프로세서의 처리 리소스에서 상기 한세트의 할당 대상인 서비스 중의 할당 대상 인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 상기 프로세서의 처리 리소스는 프로세서 코 어를 포함하며,

제1 할당 유닛은,

한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응 답 속도 임계값보다 작은 할당 대상인 서비스를 제2 운영체제에 할당하는 제1 할당 모듈;

한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당하는 제2 할당 모듈;

한세트의 할당 대상인 서비스에 민감한 정보가 포함되는 할당 대상인 서비스를 타깃 운영체제에 할당하되, 타깃 운영체제는 제1 운영체제 및 제2 운영체제에서, 사용 객체와 인터랙션하는 빈도가 낮은 운영체제인 것인 제3 할당 모듈; 중의 적어도 하나를 포함하는 것을 특징으로 하는 임베디드 시스템의 실행 장치.

청구항 19

삭제

청구항 20

삭제

청구항 21

삭제

청구항 22

삭제

청구항 23

삭제

청구항 24

삭제

청구항 25

삭제

청구항 26

삭제

청구항 27

삭제

청구항 28

삭제

청구항 29

삭제

청구항 30

삭제

청구항 31

삭제

청구항 32

삭제

청구항 33

삭제

청구항 34

삭제

청구항 35

삭제

청구항 36

삭제

청구항 37

삭제

청구항 38

삭제

청구항 39

삭제

청구항 40

삭제

청구항 41

삭제

청구항 42

삭제

청구항 43

삭제

청구항 44

삭제

청구항 45

삭제

청구항 46

삭제

청구항 47

삭제

청구항 48

삭제

청구항 49

삭제

청구항 50

삭제

청구항 51

삭제

청구항 52

삭제

청구항 53

삭제

청구항 54

삭제

청구항 55

삭제

청구항 56

삭제

청구항 57

삭제

청구항 58

삭제

청구항 59

삭제

청구항 60

삭제

청구항 61

삭제

청구항 62

삭제

청구항 63

삭제

청구항 64

삭제

청구항 65

삭제

청구항 66

삭제

청구항 67

삭제

청구항 68

삭제

청구항 69

삭제

청구항 70

삭제

청구항 71

삭제

청구항 72

삭제

청구항 73

삭제

청구항 74

삭제

청구항 75

삭제

청구항 76

삭제

청구항 77

삭제

발명의 설명

기술 분야

[0001] 본 출원의 실시예는 컴퓨터 분야에 관한 것으로, 구체적으로는 임베디드 시스템의 실행 방법 및 장치, 임베디드 시스템 및 칩에 관한 것이다.

배경 기술

[0002] 반도체 산업 및 집적 회로 기술의 신속한 발전에 따라, 멀티 코어 프로세서가 많은 분야에서의 중요한 컴퓨팅 유닛이 되었다. 응답 속도에 민감한 응용 장면에서, 프로세서에 전용 가속 하드웨어를 추가하는 방식을 통해, 멀티 코어에서 응답 속도에 민감하지 않는 서비스를 베어링하고, 가속 하드웨어에서 응답 속도에 민감한 서비스를 베어링할 수 있다. 하지만, 상기 서비스 처리 방식에 있어서, 멀티 코어 프로세서 다수의 처리 리소스(예를 들면, 핵심 리소스)가 유휴 상태이므로, 처리 리소스의 전반 이용율이 낮다.

발명의 내용

- [0003] 본 출원의 실시예는 적어도 관련 기술에 존재하는 멀티 코어 프로세서 다수의 처리 리소스가 유향 상태이므로, 처리 리소스의 전반 이용율이 낮은 기술적 과제를 해결하기 위한 임베디드 시스템의 실행 방법 및 장치, 임베디드 시스템 및 칩을 제공한다.
- [0004] 본 출원 실시예의 일 측면에 따르면, 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 상기 임베디드 시스템에서 대응되는 운영체제에 할당하되, 상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 상기 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 상기 제1 운영체제와 상기 제2 운영체제는 프로세서에서 실행되고, 상기 제1 운영체제의 응답 속도는 상기 제2 운영체제보다 빠른 것인 단계; 상기 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하되, 상기 리소스 할당 결과는 상기 프로세서의 처리 리소스에서 상기 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 상기 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것인 단계; 상기 할당 대상인 서비스 각각과 대응되는 운영체제 및 상기 리소스 할당 결과에 따라, 상기 프로세서의 처리 리소스를 상기 제1 운영체제 및 상기 제2 운영체제에 할당하는 단계; 를 포함하는 임베디드 시스템의 실행 방법을 제공한다.
- [0005] 본 출원 실시예의 다른 일 측면에 따르면, 프로세서에서 실행되는 제1 운영체제 및 제2 운영체제; 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 서비스 관리 모듈; 상기 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 리소스 동적 할당 모듈; 상기 할당 대상인 서비스 각각과 대응되는 운영체제 및 상기 리소스 할당 결과에 따라, 상기 프로세서의 처리 리소스를 상기 제1 운영체제 및 상기 제2 운영체제에 할당하는 리소스 적응형 스케줄링 모듈; 을 포함하되, 상기 제1 운영체제의 응답 속도는 상기 제2 운영체제보다 빠르고, 상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당을 수행하며, 상기 리소스 할당 결과는 상기 프로세서의 처리 리소스에서 상기 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 상기 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것인 임베디드 시스템을 더 제공한다.
- [0006] 본 출원의 또 다른 일 실시예에 따르면, 칩 및 적어도 2개의 운영체제를 포함하며, 상기 칩은 프로세서, 하드웨어 제어기, 제1 버스 및 제2 버스를 포함하되, 상기 제1 버스의 대역폭은 상기 제2 버스의 대역폭보다 높고, 또한 상기 제1 버스는 멀티 마스터 멀티 슬레이브 모드로 구성되고, 상기 제2 버스는 단일 마스터 멀티 슬레이브 모드로 구성되며, 상기 적어도 2개의 운영체제는 상기 프로세서에 기반하여 실행되고, 상기 프로세서의 처리 리소스는 상기 적어도 2개의 운영체제에 동적 할당되고, 상기 프로세서의 처리 리소스는 프로세서 코어를 포함하며, 상기 적어도 2개의 운영체제는 상기 제1 버스를 통해 통신을 수행하며; 상기 적어도 2개의 운영체제는 상기 제2 버스를 통해 상기 하드웨어 제어기에 대한 제어를 실현하는 것인 임베디드 시스템을 더 제공한다.
- [0007] 본 출원의 또 다른 측면에 따르면, 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 상기 임베디드 시스템에서 대응되는 운영체제에 할당하는 제1 할당 유닛; 상기 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 제1 결정 유닛; 상기 할당 대상인 서비스 각각과 대응되는 운영체제 및 상기 리소스 할당 결과에 따라, 상기 프로세서의 처리 리소스를 상기 제1 운영체제 및 상기 제2 운영체제에 할당하는 제2 할당 유닛; 을 포함하되, 상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 상기 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 상기 제1 운영체제와 상기 제2 운영체제는 프로세서에서 실행되고, 상기 제1 운영체제의 응답 속도는 상기 제2 운영체제보다 빠르고, 상기 리소스 할당 결과는 상기 프로세서의 처리 리소스에서 상기 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 상기 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것인 임베디드 시스템의 실행 장치를 더 제공한다.
- [0008] 본 출원 실시예의 또 다른 측면에 따르면, 프로그래밍 가능한 로직 회로 및 실행 가능한 명령어 중의 적어도 하나를 포함하고, 전자 기기에서 실행되며, 상술한 어느 한 항의 방법 실시예에서의 단계를 실현하기 위한 칩을 더 제공한다.
- [0009] 본 출원 실시예의 또 다른 측면에 따르면, 저장 유닛 및 상기 저장 유닛과 연결되는 처리 유닛을 포함하되, 상

기 저장 유닛은 프로그램을 저장하고, 상기 처리 유닛은 상기 프로그램을 실행하며, 상술한 어느 한 항에 따른 방법 실시예에서의 단계를 실행하기 위한 BMC 칩을 더 제공한다.

[0010] 본 출원 실시예의 또 다른 측면에 따르면, 적어도 하나의 프로세서; 적어도 하나의 프로그램을 저장하기 위한 적어도 하나의 메모리; 를 포함하되, 상기 적어도 하나의 프로그램이 상기 적어도 하나의 프로세서에 의해 실행될 시, 상기 적어도 하나의 프로세서는 상술한 어느 한 항에 따른 방법 실시예에서의 단계를 실행하는 메인 보드를 더 제공한다.

[0011] 본 출원 실시예의 또 다른 측면에 따르면, 프로세서, 통신 인터페이스, 메모리 및 통신 버스를 포함하는 서버로서, 프로세서, 통신 인터페이스, 메모리는 통신 버스를 통해 상호 간의 통신을 완성하고, 메모리는 컴퓨터 프로그램을 저장하며, 메모리에 저장된 프로그램이 프로세서에 의해 실행될 시, 상술한 어느 한 항에 따른 방법 실시예에서의 단계를 실행하는 서버를 더 제공한다.

[0012] 본 출원의 또 다른 실시예에 따르면, 컴퓨터 프로그램이 저장되어 있는 컴퓨터 판독가능 저장 매체로서, 상기 컴퓨터 프로그램은 실행 시 상술한 어느 한 항에 따른 방법 실시예에서의 단계를 수행하도록 구성되는 컴퓨터 판독가능 저장 매체를 더 제공한다.

[0013] 본 출원의 또 다른 실시예에 따르면, 메모리 및 프로세서를 포함하고, 상기 메모리에는 컴퓨터 프로그램이 저장되며, 상기 프로세서는 상기 컴퓨터 프로그램을 실행하여 상술한 어느 한 항에 따른 방법에서의 단계를 수행하도록 구성되는 전자 기기를 더 제공한다.

[0014] 본 실시예에서는, 프로세서의 서로 다른 처리 리소스에서 임베디드 시스템의 서로 다른 운영체제를 실행하는 방식을 사용한다. 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당하되, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 제1 운영체제와 제2 운영체제는 프로세서에서 실행되고, 제1 운영체제의 응답 속도는 제2 운영체제보다 빠르며; 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하되, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함하며; 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.

[0015] 프로세서에서는 임베디드 시스템의 적어도 2개의 운영체제가 실행되고, 서로 다른 운영체제는 응답 속도가 서로 다르므로, 응답 속도 요구사항이 서로 다른 서비스를 실행하는데 사용될 수 있다. 리소스 동적 할당 규칙에 기반하여 프로세서의 실행 대상인 서비스를 서로 다른 운영체제에 할당하고, 실행 대상인 서비스를 서로 다른 처리 리소스에 할당함으로써, 실행 대상인 서비스와 운영체제의 대응관계 및 실행 대상인 서비스와 처리 리소스의 대응관계에 기반하여 프로세서의 처리 리소스를 서로 다른 운영체제에 동적 할당할 수 있다. 상기 서비스 실행 방식을 통해, 별도의 가속 하드웨어를 증가할 필요가 없고, 동시에 모든 서비스는 모두 프로세서 내의 처리 리소스에 의해 실행되는 것으로서, 기존의 프로세서에 의해 실행되는 서비스의 기초상에서, 기존의 가속 하드웨어에 의해 실행되는 서비스를 별도로 처리해야 하므로, 프로세서에 의해 처리되는 서비스량을 증가하는 목적을 실현할 수 있고, 처리 리소스의 전반 이용율을 향상시키는 기술적 효과에 도달할 수 있으므로, 관련 기술에 존재하는 멀티 코어 프로세서 다수의 처리 리소스가 유휴 상태이므로 코어 리소스의 전반 이용율이 낮은 기술적 문제를 해결한다.

도면의 간단한 설명

[0016] 여기에서의 도면은 명세서에 병입되어 명세서의 일부분을 형성하며, 본 출원에 적합한 실시예를 도시하여, 명세서와 함께 본 출원의 원리를 해석하는데 사용된다.

이하, 본 출원의 실시예 또는 관련 기술에서의 해결수단을 더 명확하게 설명하기 위해, 실시예 또는 관련 기술에서 사용되어야 할 도면을 간단하게 설명한다. 자명한 것은, 당업자에게 있어서 창조적인 노동없이도 이러한 도면에 따라 다른 도면을 획득할 수 있다.

도 1은 본 출원의 실시예에 따른 임베디드 시스템의 실행 방법의 하드웨어 환경 예시도이다.

도 2는 본 출원의 실시예에 따른 선택적인 임베디드 시스템의 실행 방법의 흐름 예시도이다.

- 도 3은 본 출원의 실시예에 따른 선택적인 임베디드 시스템의 실행 방법의 예시도이다.
- 도 4는 본 출원의 실시예에 따른 다른 선택적인 임베디드 시스템의 실행 방법의 예시도이다.
- 도 5는 본 출원의 실시예에 따른 또 다른 선택적인 임베디드 시스템의 실행 방법의 예시도이다.
- 도 6은 본 출원의 실시예에 따른 또 다른 선택적인 임베디드 시스템의 실행 방법의 예시도이다.
- 도 7은 본 출원의 실시예에 따른 또 다른 선택적인 임베디드 시스템의 실행 방법의 예시도이다.
- 도 8은 본 출원의 실시예에 따른 다른 선택적인 임베디드 시스템의 실행 방법의 흐름 예시도이다.
- 도 9는 본 출원의 실시예에 따른 또 다른 선택적인 임베디드 시스템의 실행 방법의 흐름 예시도이다.
- 도 10은 본 출원의 실시예에 따른 선택적인 임베디드 시스템의 예시도이다.
- 도 11은 본 출원의 실시예에 따른 다른 선택적인 임베디드 시스템의 예시도이다.
- 도 12는 본 출원의 실시예에 따른 선택적인 임베디드 시스템의 실행 장치의 구조 블록도이다.

발명을 실시하기 위한 구체적인 내용

- [0017] 이하, 당업자가 본 출원의 해결수단을 더 잘 이해할 수 있도록, 본 출원 실시예에서의 도면과 함께 본 출원 실시예에서의 해결수단을 명확하고 완전하게 설명한다. 자명한 것은, 설명되는 실시예는 단지 본 출원의 일부 실시예일 뿐이며, 전부 실시예는 아니다. 본 출원에서의 실시예에 기반하여, 당업자가 창조적인 노동없이 획득한 모든 다른 실시예는 모두 본 출원의 보호범위에 속한다.
- [0018] 설명해야 할 것은, 본 출원의 명세서, 특허청구범위 및 상기 도면에서의 “제1”, “제2” 등 용어는 유사한 객체를 구분하기 위한 것으로, 특정된 순서 또는 선후 순서를 설명하기 위한 것은 아니다. 여기서 설명되는 본 출원의 실시예를 여기에 도시되거나 설명되는 순서 이외의 순서로 실시될 수 있도록, 상기와 같이 사용되는 데이터는 적절한 상황에서 호환될 수 있다는 것을 이해해야 한다. 또한, 용어 “포함”, “구비” 및 이들의 임의의 변형은 비배타적인 포함을 커버하는 것을 의미한다. 예를 들면, 일련의 단계 또는 유닛을 포함하는 프로세스, 방법, 시스템, 제품 또는 기기는 명확하게 열거되는 단계 또는 유닛에만 한정될 필요가 없으며, 명확하게 열거되지 않은 또는 상기 프로세스, 방법, 제품 또는 기기의 고유의 기타 단계 또는 유닛도 포함할 수 있다.
- [0019] 본 출원의 실시예에 의해 제공되는 방법 실시예는 서버, 컴퓨터 단말기, 기기 단말기 또는 유사한 연산 장치에서 실행될 수 있다. 서버에서 실행되는 것을 예로 들면, 도 1은 본 출원의 실시예에 따른 임베디드 시스템의 실행 방법의 하드웨어 환경 예시도이다. 도 1에 도시된 바와 같이, 서버는 하나 또는 복수(도 1에 하나만 도시됨)의 프로세서(102, 프로세서(102)는 마이크로 프로세서(MCU) 또는 프로그래밍 가능한 로직 디바이스(FPGA) 등의 처리 장치를 포함할 수 있으나 이에 한정되지 않음) 및 데이터를 저장하는 메모리(104)를 포함할 수 있으며, 예시적인 일 실시예에서, 상기 서버는 통신 기능을 위한 전송 장치(106) 및 입출력 장치(108)를 더 포함할 수 있다. 당업자가 이해할 수 있는 것은, 도 1에 도시된 구조는 예시일 뿐이며, 상기 서버의 구조를 한정하는 것은 아니다. 예를 들면, 서버는 도 1에 도시된 것보다 더 많거나 더 적은 구성요소를 더 포함할 수 있거나, 또는 도 1에 도시된 것과 같은 기능 혹은 도 1에 도시된 기능보다 더 많은 다른 구성을 가질 수 있다.
- [0020] 메모리(104)는, 예를 들면, 본 출원 실시예에서의 임베디드 시스템의 실행 방법에 대응되는 컴퓨터 프로그램과 같은 응용 소프트웨어의 소프트웨어 프로그램 및 모듈 등 컴퓨터 프로그램을 저장하는데 사용될 수 있고, 프로세서(102)는 메모리(104) 내에 저장된 컴퓨터 프로그램을 실행하여, 다양한 기능 응용 및 데이터 처리를 수행할 수 있으며, 즉 상기 방법을 실현할 수 있다. 메모리(104)는 고속 랜덤 액세스 메모리를 포함할 수 있으며, 하나 또는 복수의 자기 저장 장치, 플래시 메모리 또는 다른 비휘발성 솔리드 스테이트 메모리와 같은 비휘발성 메모리를 더 포함할 수 있다. 일부 실시예에서는, 메모리(104)는 프로세서(102)에서 원격으로 설치된 메모리를 진일보로 포함할 수 있으며, 이러한 원격 메모리는 네트워크를 통해 서버에 연결될 수 있다. 상기 네트워크의 실시예는 인터넷, 인트라넷, 근거리 통신망, 이동 통신망 및 그 조합을 포함하되, 이에 한정되지 않는다.
- [0021] 전송 장치(106)는 하나의 네트워크를 통해 데이터를 송수신한다. 상기 네트워크의 구체적인 실시예는 서버의 통신 공급업체에 의해 제공되는 무선 네트워크를 포함할 수 있다. 하나의 실시예에서는, 전송 장치(106)는 하나의 네트워크 어댑터(Network Interface Controller, NIC로 약칭함)를 포함하며, 기지국을 통해 다른 네트워크 기기와 서로 연결되어 인터넷과 통신할 수 있다. 하나의 실시예에서는, 전송 장치(106)는 무선 주파수(Radio Frequency, RF로 약칭함) 모듈일 수 있으며, 이는 무선 형태로 인터넷과 통신한다.

- [0022] 본 실시예에서는 상기 서버에 응용되는 임베디드 시스템의 실행 방법을 제공한다. 도 2는 본 출원의 실시예에 따른 선택적인 임베디드 시스템의 실행 방법의 흐름 예시도이다. 도 2에 도시된 바와 같이, 상기 흐름은 하기와 같은 단계 S202, S204 및 S206을 포함한다.
- [0023] 단계 S202에서, 리소스 동적 할당 규칙에 따라, 한 세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당한다.
- [0024] 본 실시예에서의 임베디드 시스템의 실행 방법은 프로세서의 처리 리소스에서 서비스를 실행하여 처리 리소스를 동적 밸런스 스케줄링하는 장면에 응용될 수 있다. 상기 임베디드 시스템은 임베디드 멀티 시스템일 수 있다. 임베디드 멀티 시스템은 임베디드 시스템의 멀티 코어 프로세서에서 다양한 운영체제(예를 들면, 제1 운영체제, 제2 운영체제 등)가 실행되고, 또한 이러한 운영체제가 동시에 동일한 임베디드 시스템에서 실행되는 것을 가리킨다. 다양한 운영체제는 동일한 유형의 운영체제일 수도 있고, 예를 들면, 이기종 운영체제(아키텍처가 서로 다른 서로 다른 유형의 운영체제)와 같은 서로 다른 유형의 운영체제일 수도 있다. 프로세서에서의 서비스 실행은 프로세서의 복수의 코어에서의 처리 리소스를 통해 병렬로 실행할 수 있는 것이다. 여기에서의 프로세서는, 예를 들면, 8코어 프로세서와 같은 멀티 코어 프로세서일 수도 있고, 다른 개수의 코어를 포함하는 프로세서일 수도 있다. 본 실시예에서는 멀티 코어 프로세서에 포함되는 코어 수에 대해 한정하지 않는다.
- [0025] 관련 기술에서는, 반도체 산업 및 집적 회로 기술의 신속한 발전에 따라, 멀티 코어 프로세서가 클라우드 컴퓨팅, AI(Artificial Intelligence, 인공 지능), 빅데이터, 산업 인터넷 및 5G(5th Generation Mobile Communication Technology, 5세대 이동통신 기술) 등 분야의 중요한 컴퓨팅 유닛이 되었다. 하드웨어 컴퓨팅 리소스 공유를 실현하기 위해, 가상화 기술을 도입하여 멀티 코어 프로세서의 이용율을 향상시킬 수 있다. 즉, 프로세서의 하드웨어 플랫폼에 기반하여 복수의 가상머신을 가상화하며, 가상머신 각각은 하나의 독립적인 운영체제를 실행한다.
- [0026] 하지만, 가상머신에 기반하는 운영체제는 가상머신 관리 등 오버헤드가 증가하므로, 일반적으로 서비스의 실시간성 요구사항을 충족시킬 수 없다. 따라서, 응답 속도(서비스의 실시간성과 관련됨)에 민감한 응용의 장면에서, 일반적으로는 물리머신을 독점하는 형태를 직접 적용하여 운영체제를 실행하고, 이러한 “베어메탈”식의 프로세서 사용 방식은 서비스 처리 딜레이를 현저히 감소시킬 수 있다. 임베디드 시스템을 개발하는 각 산업 분야(예를 들면, 서버, 사물인터넷 등)에서는, 기능에 대한 요구사항이 점점 높아지는 시스템 관리 기능을 실현하기 위해, 마찬가지로 이에 베어링되는 서비스를 응답 속도 요구사항(실시간성 요구사항)의 서로 다른 정도에 따라 통합 구분하고, 서로 다른 서비스 유형을 위해 서로 다른 하드웨어 실현 플랫폼을 구성한다. 예를 들면, 복잡하고 종류가 많은 비실시간형 서비스는 멀티 코어 프로세서 CPU(Central Processing Unit, 중앙 처리 장치, 즉 컴퓨터 시스템의 연산 및 제어 코어로서, CPU는 정보 처리, 프로그램 실행의 최종 실행 유닛임)에서 실행하고, 응답 속도 요구사항이 높은 서비스는 전용 가속 하드웨어에서 실현한다. 여기에서의 가속 하드웨어는 FPGA(Field Programmable Gate Array, 현장 프로그래밍 가능한 게이트 어레이), CPLD(Complex Programmable Logic Device, 콤플렉스 프로그래밍 가능한 로직 디바이스) 또는 전용 ASIC(Application Specific Integrated Circuit, 전용 집적 회로)칩 등을 포함할 수 있으나, 이에 한정되지 않는다.
- [0027] 하지만, 임베디드 시스템에서 멀티 코어 프로세서에 가속 하드웨어를 추가하는 해결수단의 경우, 프로세서의 모든 처리 리소스(예를 들면, 프로세서 코어)에서는 통상적으로 단일 종류의 운영체제만 실행하고, 또한 다수의 처리 리소스는 대부분 시간 유휴상태이므로, 처리 리소스의 전반 이용율이 낮다. 여기서, 일반적인 임베디드 제품(예를 들면, 핸드폰, 태블릿 PC)은 통상적으로 멀티 코어로 한가지 운영체제(예를 들면, 안드로이드, IOS, Windows)를 실행하고, CPU 코어의 로드는 대부분의 경우 점유율이 30% 이하(CPU 능력은 통상적으로 과잉임)이며, CPU는 타임 슬라이스에 따라 작동되므로, 점유되지 않을 경우 유휴상태이다. 또한, 응답 시간에 민감한 일부분의 서비스에 대응하기 위해 FPGA/CPLD 등 가속 하드웨어 리소스 투입을 증가하는 것은 비용 제어에 유리하지 않다.
- [0028] 상기 기술적 과제의 적어도 일부분을 해결하기 위한 본 실시예에서는, 프로세서의 서로 다른 처리 리소스에서 임베디드 시스템의 서로 다른 운영체제를 실행하는 방식을 사용한다. 서로 다른 운영체제는 응답 속도가 서로 다르므로, 응답 속도 요구사항이 서로 다른 서비스를 실행하는데 사용될 수 있다. 리소스 동적 할당 규칙에 기반하여 프로세서의 실행 대상인 서비스를 서로 다른 운영체제에 할당하고, 실행 대상인 서비스를 서로 다른 처리 리소스에 할당함으로써, 실행 대상인 서비스와 운영체제의 대응관계 및 실행 대상인 서비스와 처리 리소스의 대응관계에 기반하여 프로세서의 처리 리소스를 서로 다른 운영체제에 동적 할당할 수 있다. 상기 서비스 실행 방식을 통해, 별도의 가속 하드웨어를 증가할 필요가 없으므로, 비용 제어가 용이하고, 동시에 모든 서비스는

모두 프로세서 내의 처리 리소스에 의해 실행되는 것으로서, 기존의 프로세서에 의해 실행되는 서비스의 기초상에서, 기존의 가속 하드웨어에 의해 실행되는 서비스를 별도로 처리해야 하므로, 프로세서에 의해 처리되는 서비스량이 증가되고, 처리 리소스의 전반 이용율이 향상될 수 있다.

[0029] 본 실시예에서는, 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 제1 운영체제의 응답 속도는 제2 운영체제의 응답 속도보다 빠르다. 제1 운영체제와 제2 운영체제는 프로세서에서 실행되며, 즉 프로세서의 서로 다른 처리 리소스에서 실행되고, 프로세서의 처리 리소스는 제1 운영체제의 처리 리소스(제1 운영체제에 스케줄링된 처리 리소스), 제2 운영체제의 처리 리소스(제2 운영체제에 스케줄링된 처리 리소스) 및 할당되지 않은 처리 리소스 중의 적어도 하나를 포함할 수 있다. 여기서, 프로세서는 멀티 코어 프로세서(예를 들면, 8코어 프로세서)일 수 있으며, 프로세서의 처리 리소스(즉, 프로세서 리소스)는 프로세서 코어(프로세서 하드 코어)를 포함할 수도 있고, 예를 들면, 제어기 로직 유닛, 외부 인터페이스 및 버스 리소스 등과 같은 다른 유형의 리소스를 포함할 수도 있다.

[0030] 프로세서가 실행되는 과정에서, 한세트의 할당 대상인 서비스, 즉 제1 운영체제 및 제2 운영체제에 할당 대상인 서비스를 획득할 수 있다. 서로 다른 할당 대상인 서비스는 응답 속도, 서비스 리소스 점유율, 다른 서비스와의 서비스 결합도 및 서비스 중요성 등 차원에서 구별점이 존재할 수 있다. 따라서, 리소스 동적 할당 규칙을 사전에 구성할 수 있다. 리소스 동적 할당 규칙은 서비스 할당을 위한 규칙을 포함할 수 있으며, 대응되는 운영체제의 처리 리소스에서 자체에 할당된 서비스를 실행하도록, 서비스를 대응되는 운영체제에 할당한다. 선택적으로, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함할 수 있다. 서로 다른 할당 규칙은 대응되는 우선순위를 갖는다. 예를 들면, 우선순위가 높은 것에서 낮은 것의 순서에 따라 서비스 중요성, 서비스 결합도, 서비스 응답 속도 및 서비스 리소스 점유율이다. 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스(또는 할당될 태스크, 서로 다른 할당 대상인 서비스는 서로 다른 프로세스에 대응될 수 있음)을 임베디드 시스템에서 대응되는 운영체제에 할당하여, 서비스 할당 결과를 획득할 수 있다.

[0031] 선택적으로, 응답 시간에 대한 제약에 기반하여, 제1 운영체제는 명확하게 고정된 시간 제약을 갖는 운영체제일 수 있으며, 모든 처리 과정(태스크 스케줄링)은 고정된 시간 제약 내에 완성되어야 하며, 그렇지 않을 경우 시스템에 오류가 발생한다. 상기 제1 운영체제는, 예를 들면, FreeRTOS, RTLinux 등과 같은 실시간 운영체제(Real Time Operating System, RTOS로 약칭함)일 수도 있고, 다른 임베디드 시스템에서의 실시간 운영체제일 수도 있다. 제2 운영체제는 상기 특성을 구비하지 않는다. 제2 운영체제는 일반적으로 공평 태스크 스케줄링 알고리즘을 적용하고, 스레드/프로세스의 개수가 증가될 경우, CPU 시간을 공유해야 하며, 태스크 디버깅은 불안정성을 가진다. 상기 제2 운영체제는, 예를 들면, contiki, Helios 및 Linux(스펠링은 GNU/Linux로서, 한세트의 자유로 전파될 수 있는 Unix 운영체제임)과 같은 비실시간 운영체제일 수도 있고, 다른 임베디드 시스템에서의 비실시간 운영체제일 수도 있다. 여기서, Linux 시스템은 POSIX(Portable Operating System Interface, 이식 가능한 운영체제 인터페이스)에 기반하는 멀티 사용자, 멀티 태스크, 멀티 스레드와 멀티 CPU를 지원하는 운영체제이다.

[0032] 이와 대응되게, 제1 운영체제에 할당되는 서비스는 통상적으로 실시간성 서비스고, 실시간성 서비스는 소정의 시간 내에 스케줄링 받아야 할 서비스를 가리키며, 상기 서비스는 프로세서에 의해 충분히 신속한 속도로 처리되어야 하고, 그 처리 결과는 소정의 시간 내에 생산 과정을 제어하거나 처리 시스템에 대해 신속하게 응답할 수 있어야 한다. 전형적인 장면으로서, 산업 제어 과정에서 로봇암에 대한 제어는 실시간성 서비스에 속하고, 시스템은 로봇암의 오작동이 검출되기 전에 적시로 조치를 취해야 하며, 그렇지 않을 경우 심각한 후과를 초래할 수 있다. 제2 운영체제에 할당되는 서비스는 통상적으로 비실시간성 서비스고, 비실시간성 서비스는, 예를 들면, 서버에서 온도 센서(sensor)의 센서 데이터를 읽는 것과 같은 스케줄링 시간에 민감하지 않는 서비스를 가리키며, 스케줄링의 지연에 대해 소정의 허용도를 갖는다.

[0033] 설명해야 할 것은, 실시간 운영체제는 외부 이벤트 또는 데이터가 생성될 경우, 수신하여 충분히 신속한 속도로 처리하는 것을 가리킨다. 그 처리 결과는 소정의 시간 내에 생산 과정을 제어하거나 처리 시스템에 대해 신속하게 응답할 수 있고, 이용할 수 있는 모든 리소스를 스케줄링하여 실시간 서비스를 완성하며, 모든 실시간성 서비스가 조정일치로 실행되도록 제어하는 운영체제는 적시 응답 및 높은 신뢰도의 특성을 갖는다.

[0034] 선택적으로, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 단계는 서비스 관리 모듈에 의해 실행될 수 있다. 도 3에 도시된 바와 같이, 상기 서비스 관리 모듈은 제1 운영체제 또는 제2 운영체제에서 실행되는 소프트웨어 모듈일 수 있다. 제2 운영체제에서 실행되는 것을 예로 들면, 서비스 관리 모듈은 Linux 시스템

에서의 소프트웨어로 실현할 수 있다. 상기 서비스 모듈은 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당할 수 있다.

- [0035] 단계 S204에서, 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정한다. 여기서, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이다
- [0036] 할당 대상인 서비스 각각을 대응되는 운영체제에 할당한 다음, 서비스 할당 결과에 따라, 할당 대상인 서비스 각각에 대응되는 처리 리소스를 할당하여, 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 획득할 수 있다. 할당 대상인 서비스에 처리 리소스를 할당할 경우, 제1 운영체제에 할당된 서비스에 제1 운영체제의 처리 리소스를 할당하고, 제2 운영체제에 할당된 서비스에 제2 운영체제의 처리 리소스를 할당하며, 동시에 로드 밸런스를 고려하여, 할당되지 않은 처리 리소스가 존재할 경우, 일부분 서비스에 할당되지 않은 처리 리소스를 할당할 수 있다.
- [0037] 프로세서의 처리 리소스는 타임 슬라이스를 단위로 처리 리소스의 동적 할당을 수행할 수 있으며, 처리 리소스가 속하는 운영체제 및 서비스를 빈번하게 전환하므로 처리 시간이 타임 슬라이스의 정수배가 아닐 수 있음을 고려하여, 일부분 서비스의 응답 시간이 연장되고, 프로세서 코어를 단위로 제1 운영체제와 제2 운영체제에 할당될 수 있으며, 즉 프로세서의 프로세서 코어는 전반 프로세서 코어를 단위로 대응되는 운영체제에 할당되고, 운영체제 각각에 할당되는 프로세서 코어의 개수는 정수개이며, 서로 다른 운영체제에 할당되는 프로세서 코어는 서로 다르다.
- [0038] 선택적으로, 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과는 리소스 동적 할당 물들에 의해 실행될 수 있는 것으로 결정한다. 도 3에 도시된 바와 같이, 상기 리소스 동적 할당 모듈은 제1 운영체제 또는 제2 운영체제에서 실행되는 소프트웨어 모듈일 수 있다. 제2 운영체제에서 실행되는 것을 예로 들면, 리소스 동적 할당 모듈은 제2 운영체제에서의 소프트웨어 모듈로 실행할 수 있으며, 이는 서비스 관리 모듈의 출력에 따라, 서비스의 처리 리소스에 대한 동적 할당을 수행할 수 있다. 여기서, 소프트웨어 모듈은 사전 설정 기능을 갖는 프로그램 모듈일 수 있다. 예를 들면, 리소스 동적 할당 모듈의 경우, 이는 리소스 동적 할당 기능을 갖는 프로그램 모듈일 수 있고, 서비스 관리 모듈의 경우, 이는 서비스 관리 기능을 갖는 프로그램 모듈일 수 있다. 소프트웨어 모듈은 전체로서 배치 및 조정될 수 있으며, 동시에 서로 다른 응용 엔지니어링에 응용될 수 있다.
- [0039] 단계 S206에서, 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.
- [0040] 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당할 수 있다. 선택적으로, 프로세서의 할당되지 않은 처리 리소스를 이와 대응되는 운영체제에 할당할 수 있다. 할당되지 않은 처리 리소스는 할당되지 않은 처리 리소스와 할당 대상인 서비스의 대응관계 및 할당 대상인 서비스와 운영체제의 대응관계에 기반하여 결정되는 것일 수 있다.
- [0041] 선택적으로, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 단계는 리소스 적응형 스케줄링(resource adaptive scheduling) 모듈(예를 들면, 코어 적응형 스케줄링 모듈)에 의해 실행될 수 있다. 상기 리소스 적응형 스케줄링 모듈은 제1 운영체제 또는 제2 운영체제에서 실행되는 소프트웨어 모듈일 수 있다. 제2 운영체제에서 실행되는 것을 예로 들면, 리소스 적응형 스케줄링 모듈은 Linux 시스템에서의 소프트웨어로 실현할 수 있다. 이는 서비스 관리 모듈의 출력 및 리소스 동적 할당 모듈의 출력에 따라, 프로세서의 처리 리소스(예를 들면, 프로세서 하드 코어 리소스)에 대한 실제 스케줄링 동작을 완성할 수 있다. 도 4에 도시된 바와 같이, 코어 리소스 적응형 모듈의 리소스 스케줄링을 거쳐, (M+N)개 코어 중의 M개 코어는 실시간 운영체제에 스케줄링되고, N개 코어는 비실시간 운영체제에 스케줄링된다.
- [0042] 예를 들면, 전반 프로세서 시스템이 실시간 및 비실시간 서비스의 병행 처리 능력을 갖도록, 동일한 프로세서의 서로 다른 하드 코어에서 이기종의 운영체제(이기종 운영체제)를 실행할 수 있고, 동시에 서로 다른 운영체제에 의해 점유되는 프로세서 하드 코어 리소스(예를 들면, 프로세서 코어)에 대한 적응형 조정을 통해, 프로세서 리소스 이용율의 현저한 향상을 실현할 수 있다. 여기서, 이기종은 임베디드 시스템의 동일한 멀티 코어 프로세서에서 실행되는 운영체제 유형이 서로 다른 것을 가리키고, 멀티 시스템은 임베디드 시스템의 동일한 멀티 코어 프로세서에서 실행되는 운영체제 개수가 복수 개인 것을 가리키며, 또한 이러한 운영체제는 시간 차원에서 동시에 실행된다.
- [0043] 여기에서 설명해야 할 것은, 서버의 경우, 서버는 적어도 확장 가능성이 높고 안정성이 높은 특성을 갖는다. 여

기서, 엔터프라이즈 네트워크는 장기적으로 지속될 수 없으며, 네트워크 정보화의 오늘날, 서버에 소정의 확장 가능성이 없으면, 기업의 서버 사용에 영향을 미치고, 나아가 기업의 발전에 영향을 미칠수 있으므로, 확장 가능성은 서버에 필요한 가장 기본적인 특성이고, 높은 확장 가능성을 가져야만 이후의 더 우수한 사용을 보장할 수 있다. 확장 가능성은 하드웨어에서의 확장 가능성 이외에, 소프트웨어에서의 확장 가능성을 더 포함한다. 컴퓨터와 비교할 경우, 서버의 기능은 여전히 복잡하므로, 하드웨어 구성 측면에서 뿐만아니라, 소프트웨어 구성 도 아주 중요하다. 더 많은 기능을 실현하려할 경우, 전수 소프트웨어 지원이 없으면 상상할 수도 없다.

[0044] 또한, 서버는 서비스의 지속적인 실행을 지원하기 위해, 대량의 데이터를 처리해야 함으로, 서버는, 예를 들면, 안정성이 높은 것과 같은 아주 중요한 특성을 더 갖는다. 서버의 데이터 전송이 안정적으로 실행되지 못할 경우, 서비스 실행에 아주 큰 영향을 미칠 수 있다.

[0045] 본 출원의 해결수단에서는 서버의 가확정성이 높은 특성을 이용하여, 하드웨어 인터페이스 신호를 생성하기 위한 제1 운영체제 및 제2 운영체제의 듀얼 소프트웨어 시스템을 도입한다. 동시에, 하드웨어 인터페이스 신호의 전송 전압을 조정하고, 서버 내부의 다른 디바이스의 실행 상태를 모니터링하기 위한 GPLD, BMC 칩 등 하드웨어 기기를 더 도입할 수 있다. 이 기초상에서, 본 출원에서는 제1 운영체제에서 요청 명령에 대응되는 하드웨어 인터페이스 신호를 생성하는 방식을 적용한다. 먼저 제1 운영체제를 통해 요청 명령을 획득하고, 다음으로 요청 명령에 대응되는 복수의 로직비트 정보를 결정하며, 마지막으로 복수의 로직비트 정보 및 타이머에 따라 요청 명령에 대응되는 하드웨어 인터페이스 신호를 생성한다. 상기 내용에서 알 수 있다 싶이, 본 출원은 제1 운영체제를 통해 요청 명령에 대응되는 하드웨어 인터페이스 신호를 생성함으로써, 소프트웨어 방식을 사용하여 하드웨어 인터페이스 신호를 시뮬레이션하여 생성하는 기술적 효과를 실현하고, 나아가 칩 자체에 관련되는 하드웨어 인터페이스 신호가 구비될 필요가 없는 하드웨어 로직 디자인의 목적에 달성하여, 칩의 디자인 난이도를 낮출 수 있을 뿐만 아니라, 칩의 디자인 비용도 감소시킬 수 있다. 본 출원은 칩에 대해 하드웨어 인터페이스 신호의 하드웨어 로직 디자인을 수행할 필요가 없는 기초상에서 소프트웨어 시스템을 이용하여 하드웨어 인터페이스 신호를 생성하는 목적에 달성함으로써, 칩의 디자인 난이도를 낮추고, 나아가 관련 기술에서 칩 자체에 제어기가 구비되어야 하는 하드웨어 로직 디자인으로 인한 칩의 디자인 비용이 높은 기술적 과제를 해결한다.

[0046] 또한, 제1 운영체제 및 제2 운영체제의 듀얼 소프트웨어 시스템을 도입하여, 서버의 안정성을 확보할 수도 있다. 제2 운영체제의 서비스 응답 속도는 제1 운영체제의 서비스 응답 속도보다 늦으므로, 서비스 응답 속도가 더 빠른 제1 운영체제를 사용하여 하드웨어 인터페이스 신호를 생성함으로써, 하드웨어 인터페이스 신호의 생성이 중단되지 않도록 보장할 수 있으며, 따라서 하드웨어 인터페이스 신호가 지속적이고 안정되게 출력하도록 확보할 수 있다.

[0047] 상기 단계, 즉 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당되되, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 제1 운영체제와 제2 운영체제는 프로세서에서 실행되고, 제1 운영체제의 응답 속도는 제2 운영체제보다 빠른 것인 단계; 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정되되, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것인 단계; 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 단계; 를 통해, 관련 기술에 존재하는 멀티 코어 프로세서 다수의 처리 리소스가 유휴 상태인 것으로 인한 코어 리소스의 전반 이용율이 낮은 과제를 해결함으로써, 처리 리소스의 이용율을 향상시킨다.

[0048] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S11을 더 포함한다.

[0049] S11에서, 규칙 구성 파일을 읽고, 리소스 동적 할당 규칙을 기록하기 위한 규칙 구조체를 생성한다.

[0050] 리소스 동적 할당 규칙은 규칙 구성 파일에 기반하여 구성되는 것일 수 있다. 규칙 구성 파일을 읽고, 리소스 동적 할당 규칙을 기록하기 위한 규칙 구조체를 생성할 수 있다. 여기서, 규칙 구성 파일은 로드 밸런스 전략 파일(payload_balance.config)일 수 있으며, 로드 밸런스 전략 파일은 실행되는 다양한 서비스(또는 프로세스)의 분류 방법, 실시간성 등급의 평가 원칙 등을 구성하는데 사용될 수 있다. 로드 밸런스 전략 파일에서는 서로 다른 파라미터로 리소스 동적 할당 규칙을 구성할 수 있다. 로드 밸런스 전략 구성 파일의 예시는 아래와 같다.

[0051] classification kinds =2 //의 값이 1일 경우는 중요 및 비중요 등 속성에 따라 프로세스를 분류하는 것을 나타

내고, 그렇지 않을 경우는 사전 설정된 분류 방법(예를 들면, 실시간과 비실시간)에 따라 프로세스를 분류하는 것을 나타낸다.

- [0052] real-time grade evaluation=2 //의 값이 1일 경우는 지나간 통계 분(statistic minutes) 내 CPU의 평균 점유율을 프로세스 실시간성 등급 평가 원칙으로 하는 것을 나타내고, 그렇지 않을 경우는 사전 설정된 우선순위를 프로세스 실시간성 등급 평가 원칙으로 하는 것을 나타낸다.
- [0053] statistic minutes =5 //는 각 프로세스의 평균 점유율의 통계 시간(단위가 minute, 분임)을 나타내고, real-time grade evaluation이 1일 경우 유효하다.
- [0054] 선택적으로, 리소스 동적 할당 규칙은 로드 밸런스 전략 모듈에 저장될 수 있다. 여기서, 로드 밸런스 전략 모듈은 제1 운영체제 또는 제2 운영체제에서 실행되는 소프트웨어 모듈(예를 들면, Linux 시스템에서 실행되는 소프트웨어 모듈)일 수 있으며, 이는 서비스 관리 모듈을 위해 전략 지침을 제공할 수 있으며, 시스템에서 실행되는 다양한 서비스(또는 프로세스)의 분류 방법 및 실시간성 등급의 평가 원칙 등을 포함할 수 있다. 서비스 관리 모듈은 시스템에서의 서비스에 대해 실시간성 등급에 따라 서비스 구분과 관리를 수행할 수 있으며, 진일보로 리소스 적응형 스케줄링 모듈이 프로세서 리소스의 재할당을 수행하도록 지도할 수 있다. 예시적으로, 이는 로드 밸런스 전략 모듈에 따라 출력할 수 있고, 서비스의 실제 분류를 실행할 수 있으며, 실시간 서비스와 비실시간 서비스를 포함하는 리스트를 생성할 수 있다.
- [0055] 설명해야 할 것은, 상기 분류 방법과 실시간성 등급 평가 원칙은 오픈된 것으로서, 사용자는 어떠한 방법 또는 원칙을 사용자 정의할 수 있고, 서비스 관리 모듈이 서비스 관리를 수행함에 있어서 기반으로 하는 규칙은 동적으로 구성된 것일 수 있으며, 기존 규칙의 기초상에서 진일보로 규칙에 대한 설정을 수행할 수 있다. 서비스 관리 모듈에는 동일한 기능의 복수 개 규칙이 설정될 수 있지만, 규칙 간에는 모순이 존재하지 않는다. 즉, 규칙의 구성 시간, 규칙의 우선순위 등 규칙에 기반하여 조건을 선택하여, 작용이 동일한 규칙에서 현재 사용되는 규칙을 결정함으로써, 규칙 간에 모순이 발생하는 것을 방지할 수 있다. 상기 구성 파일 load_balance.config은 한가지 가능한 상황을 설명한다. 구성 파일에서, classification_kinds 변수는 구체적인 분류 기준(예를 들면, 서비스의 중요성 또는 실시간성에 근거함) 및 분류 유형(예를 들면, 중요한 서비스와 일반적인 서비스, 실시간 서비스와 비실시간 서비스 등)을 지시하며, real-time_grade_evaluation 변수는 실시간성 평가 기준(지나간 statistic_minutes 분 내 CPU의 평균 점유율 또는 사전 설정된 서비스 우선순위에 근거할 수 있음)을 지시하고, 실시간성 등급 유형은 사용자에게 의해 정의되며, 고, 일반, 저의 3가지로 정의될 수도 있고, 더 다양하게 세분화될 수도 있다.
- [0056] 로드 밸런스 전략 모듈의 출력은 바로 구성된 분류 방법 및 실시간성 등급 평가 원칙 등이며, 소프트웨어에 의해 실현될 시에는, 구체적인 구성 파일(예를 들면, load_balance.config 파일)일 수도 있고, 구조체 변수일 수도 있다. 이러한 파일 또는 구조체 변수는 최종적으로 모두 서비스 관리 모듈에 의해 액세스되어, 로드 밸런스의 구체적인 전략을 획득할 수 있다.
- [0057] 본 실시예에서, 규칙 구성 파일을 읽고, 리소스 동적 할당 규칙을 기록하기 위한 규칙 구조체를 생성하는 단계를 통해, 정보 구성의 편의성을 향상시킬 수 있다.
- [0058] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S21 및 S22를 더 포함한다.
- [0059] S21에서, 제2 운영체제의 외부 인터페이스를 통해 규칙 업데이트 구성 파일을 획득한다. 여기서, 규칙 업데이트 구성 파일은 구성된 리소스 동적 할당 규칙을 업데이트하기 위한 것이다.
- [0060] S22에서, 규칙 업데이트 구성 파일을 사용하여 규칙 구조체를 업데이트하여, 규칙 구조체에 기록된 리소스 동적 할당 규칙을 업데이트한다.
- [0061] 규칙 구조체는 임베디드 시스템이 실행되는 과정에서 수정이 허용되지 않는 고정 포맷일 수도 있고, 특정 포맷의 구성 파일을 통해 구성 교정을 수행할 수 있는 원활하게 구성 가능한 포맷일 수도 있다. 본 실시예에서는, 규칙 업데이트 구성 파일을 획득할 수 있되, 상기 규칙 업데이트 구성 파일은 구성된 리소스 동적 할당 규칙을 업데이트하기 위한 것이며, 규칙 업데이트 구성 파일을 사용하여 규칙 구조체를 업데이트하여, 규칙 구조체에 기록된 리소스 동적 할당 규칙을 업데이트할 수 있다.
- [0062] 규칙 업데이트 구성 파일을 사용하여 규칙 구조체를 업데이트할 경우, 직접 규칙 업데이트 구성 파일에 따라 새로운 규칙 구조체를 생성하여, 새로 생성된 규칙 구조체를 사용하여 기존의 규칙 구조체를 교체할 수도 있고, 규칙 업데이트 구성 파일이 지시하는 규칙 파라미터의 파라미터값을 사용하여 규칙 구조체에서 대응되는 규칙

파라미터의 파라미터값을 업데이트할 수도 있다.

- [0063] 선택적으로, 특정 포맷의 구성 파일은 제1 운영체제 또는 제2 운영체제의 외부 인터페이스를 통해 읽는 것일 수 있다. 처리해야 할 서비스량 정도를 고려하여, 임베디드 시스템의 리소스 동적 스케줄링 등 동작은 제2 운영체제에서 담당할 수 있다. 규칙 업데이트 구성 파일을 획득할 경우, 제2 운영체제의 외부 인터페이스를 통해 규칙 업데이트 구성 파일을 획득할 수 있다.
- [0064] 예를 들면, 로드 밸런스 전략 모듈은 고정 포맷일 수도 있고, Linux 시스템의 외부 인터페이스를 통해 구성될 수도 있다. 예를 들면, 전술한 바와 같은 특정 포맷의 구성 파일(load_balance.config)은 파일 읽기/쓰기 방식을 통해 구성 교정을 수행하는 것으로 정의될 수 있다.
- [0065] 설명해야 할 것은, 외부 인터페이스는 멀티 코어 프로세서의 외부 인터페이스로서, 네트워크 인터페이스, SPI(Serial Peripheral Interface, 직렬 외부 인터페이스) 제어기 인터페이스, UART(Universal Asynchronous Receiver/Transmitter, 범용 비동기화 송수신기) 직렬포트 등 일 수 있으며, 외부로부터 데이터를 획득할 수 있는 채널이면 된다. 파일을 읽는데 사용되는 하드웨어와 구체적인 파일 위치는 서로 다른 실현 방안이 있다. 예를 들면, 네트워크 인터페이스를 통해 Web(World Wide Web, 월드 와이드 웹) 화면으로부터 구성 파일을 로딩할 수 있고, SPI 제어기를 통해 보드카드의 SPI Flash(플래시 메모리)로부터 구성 파일을 읽을 수 있으며, UART 직렬포트를 통해 다른 PC(Personal Computer, 개인용 컴퓨터)에서의 직렬포트 데이터 송수신 소프트웨어 툴로부터 구성 파일을 획득할 수 있다.
- [0066] 본 실시예에서, 규칙 업데이트 구성 파일을 획득하고, 획득한 규칙 업데이트 구성 파일을 사용하여 규칙 구조체를 업데이트하는 단계를 통해, 리소스 동적 할당 규칙 구성의 원활성을 향상시킬 수 있다.
- [0067] 예시적인 일 실시예에서는, 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당하는 단계는 하기와 같은 단계 S31를 포함한다.
- [0068] S31에서, 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0069] 할당 대상인 서비스를 할당할 경우, 할당 대상인 서비스의 서비스 응답 속도 요구사항에 기반하여 할당 대상인 서비스를 대응되는 운영체제에 할당할 수 있다. 서비스 응답 속도는 서비스의 실시간성 등급을 평가하는데 사용될 수 있다. 서비스 응답 속도 요구사항이 높을 수록, 운영체제의 스케줄링 시간 및 응답 속도에 대해 더 민감하고, 실시간성 등급이 더 높다. 서비스 응답 속도 요구사항이 높은 서비스는 운영체제에서 충분히 빠른 속도로 처리해야 하며, 그 처리 결과는 소정의 시간 내에 생산 과정을 제어하거나 처리 시스템에 대해 신속하게 응답할 수 있고, 서비스 응답 속도 요구사항이 높지 않은 서비스는 스케줄링의 지연에 대해 소정의 허용도를 갖는다.
- [0070] 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스에 있어서, 운영체제의 스케줄링 시간 및 응답 속도에 대해 민감할 경우, 이와 같은 할당 대상인 서비스를 제1 운영체제에 할당할 수 있다(예를 들면, 실시간 서비스를 실시간 운영체제에 할당함). 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스에 있어서, 응답 속도 및 스케줄링 시간에 민감하지 않으면, 이와 같은 할당 대상인 서비스를 제2 운영체제에 할당할 수 있다(예를 들면, 비실시간 서비스를 비실시간 운영체제에 할당함). 여기서, 서비스 응답 속도 요구사항은 서비스 응답 속도의 지시 파라미터를 통해 지시되는 것일 수 있고, 설정된 응답 속도 임계값은, 예를 들면, 100ms, 200ms 및 1s 등등과 같은 밀리초 레벨의 응답 속도 임계값 또는 초 레벨의 응답 속도 임계값일 수 있으며, 본 실시예에서는 설정된 응답 속도 임계값에 대해 한정하지 않는다.
- [0071] 선택적으로, 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당할 경우, 제1 운영체제와 대응되는 제1 서비스 리스트 및 제2 운영체제와 대응되는 제2 서비스 리스트를 출력할 수 있다. 제1 서비스 리스트는 제1 운영체제에 할당되는 서비스를 기록하기 위한 것이고, 제2 서비스 리스트는 제2 운영체제에 할당되는 서비스를 기록하기 위한 것이다. 즉, 서비스 할당 결과는 제1 서비스 리스트 및 제2 서비스 리스트를 포함하고, 출력되는 제1 서비스 리스트 및 제2 서비스 리스트는 프로세서의 처리 리소스의 동적 스케줄링을 수행하는데 사용될 수 있다.
- [0072] 예를 들면, 시스템 서비스에 대해 실시간성 등급 구분을 수행하여, 실시간성 서비스와 비실시간성 서비스 리스트를 획득한다. 총 20개의 서비스가 존재한다고 가정할 경우, 그 중 실시간성 서비스는 서비스1 및 서비스2이고, 비실시간성 서비스는 서비스3 내지 서비스20이다.

- [0073] 여기서, 서비스 관리 모듈은 현재의 실행 대상인 서비스에 대해 분류를 수행한다. BMC 시스템이 처음 실행될 시, 시스템에서 현재 실행될 모든 서비스가 시스템에 있어서 알려진 것이므로, 서비스 관리 모듈은 로드 밸런싱 모듈의 출력에 따라 이러한 서비스에 대해 1차 분류를 수행한다. 분류 후의 서로 다른 서비스는 서로 다른 운영체제(RTOS 시스템과 Linux 시스템)에 할당되어 실행된다. 후속적인 실행 과정에서, 서비스 프로세스에 개수 변동(예를 들면, 프로세스 중단 또는 새로운 프로세스 시작)이 발생할 경우, 서비스 관리 모듈은 계속 서비스 구분을 수행하고, 실시간적으로 기존 서비스에 대해 로드 밸런싱 전략에 따라 구분과 관리를 수행한다. 서비스 관리 모듈은 Linux 시스템에서의 하나의 상주 프로세스일 수 있으며, 그 자체는 줄곧 실행되고 있는 것이고, 또한 현재 실행되는 프로세스에 대해 관리와 구분을 수행한다.
- [0074] 본 실시예에서, 서비스 응답 속도 요구사항에 따라 할당 대상인 서비스를 대응되는 운영체제에 할당하는 단계를 통해, 스케줄링 시간에 민감한 서비스의 서비스 응답 적시성을 보장할 수 있다.
- [0075] 예시적인 일 실시예에서는, 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당하는 단계는 하기와 같은 단계 S41을 포함한다.
- [0076] S41에서, 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0077] 할당 대상인 서비스를 할당할 경우, 할당 대상인 서비스의 서비스 리소스 점유율에 기반하여 할당 대상인 서비스를 대응되는 운영체제에 할당할 수 있다. 서비스 리소스 점유율은 단위 시간 내에 서비스가 처리 리소스에 대한 평균 점유율(예를 들면, 분당 CPU 점유율)일 수 있다. 서비스 리소스 점유율의 높고 낮음은 본 서비스의 응답 속도 및 후속 서비스의 응답 속도에 영향주므로, 서비스 리소스 점유율에 기반하여 서비스의 실시간성 등급을 평가할 수 있다. 서비스 리소스 점유율이 높을 수록, 운영체제의 스케줄링 시간 및 응답 속도에 대한 영향이 더 크고, 실시간성 등급이 더 낮으며, 서비스 리소스 점유율이 높지 않은 서비스는 운영체제의 스케줄링 시간 및 응답 속도에 대한 영향이 크지 않고, 실시간성 등급이 더 높다.
- [0078] 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당될 트랜잭션의 경우, 운영체제의 스케줄링 시간 및 응답 속도에 대한 영향이 크지 않으므로, 이와 같은 할당 대상인 서비스를 제1 운영체제에 할당할 수 있다. 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당될 트랜잭션의 경우, 운영체제의 스케줄링 시간 및 응답 속도에 대한 영향이 크므로, 이와 같은 할당 대상인 서비스를 제2 운영체제에 할당할 수 있다. 여기서, 제1 점유율 임계값은 수요에 따라 구성될 수 있으며, 이는 10%, 15%, 20% 또는 다른 임계값일 수 있다. 동시에, 상기 제1 점유율 임계값은 동적 조정될 수 있다.
- [0079] 본 실시예에서, 서비스 리소스 점유율에 따라 할당 대상인 서비스를 대응되는 운영체제에 할당하는 단계를 통해, 서비스 리소스 점유율이 낮은 트랜잭션 응답의 적시성을 보장할 수 있다.
- [0080] 예시적인 일 실시예에서는, 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당하는 단계는 하기와 같은 단계 S51 및 S52 중의 적어도 한단계를 포함한다.
- [0081] S51에서, 한세트의 할당 대상인 서비스에서 제1 운영체제의 할당된 서비스와의 서비스 결합도가 제1 결합도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당한다.
- [0082] S52에서, 한세트의 할당 대상인 서비스에서 제2 운영체제의 할당된 서비스와의 서비스 결합도가 제2 결합도 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0083] 할당 대상인 서비스를 할당할 경우, 할당 대상인 서비스의 서비스 결합도에 기반하여 할당 대상인 서비스를 대응되는 운영체제에 할당할 수 있다. 서비스 결합도는 할당 대상인 서비스와 각 운영체제에서의 할당된 서비스 간의 연관 정도를 나타내는데 사용될 수 있다. 하나의 할당 대상인 서비스와 어느 한 운영체제의 할당된 서비스의 서비스 결합도가 높을 경우, 이를 다른 운영체제에 할당하는 것이 적합하지 않다. 따라서, 할당 대상인 서비스와 각 운영체제에서의 할당된 서비스 간의 서비스 결합도에 기반하여, 할당 대상인 서비스를 대응되는 운영체제에 할당할 수 있다.
- [0084] 선택적으로, 서비스의 입력과 출력 사이의 연관관계를 통해 서비스 결합도를 평가할 수 있다. 서비스 결합도는 서로 다른 결합도 등급을 통해 나타낼 수 있다. 예를 들면, 서비스의 입력과 출력 사이에 관계가 없을 경우, 결합도 등급은 저급(또는 서비스 사이에 연관이 없는 다른 결합도 등급)이고, 하나의 서비스의 실행이 다른 응용의 출력(상기 출력이 없을 경우 입력으로서 서비스를 시작할 수 없음)에 의존할 경우, 서비스 사이의 결합도 등

급은 고급이며, 하나의 서비스의 실행에 다른 응용의 출력이 사용되지만 상기 출력이 서비스의 정상적인 실행(서비스가 대응되는 동작까지 실행될 경우 상기 출력을 획득할 수 있으면 되고, 또한 대응되는 동작은 핵심 동작이 아님)에 방해가 되지 않을 경우, 서비스 사이의 결합도 등급은 중급이다. 또한, 수치로 서비스 결합도를 나타낼 수도 있고, 한가지 또는 여러가지 결합도 조건(예를 들면, 입력과 출력 사이의 연관관계)을 통해 서비스 결합도를 평가할 수 있으며, 충족되는 결합도 조건에 대응되는 수치를 서비스 결합도의 수치로 결정한다.

[0085] 한세트의 할당 대상인 서비스에 제1 운영체제의 할당된 서비스와의 서비스 결합도가 제1 결합도 임계값 이상인 할당 대상인 서비스가 존재할 경우, 이와 같은 할당 대상인 서비스를 제1 운영체제에 할당할 수 있고, 한세트의 할당 대상인 서비스에 제2 운영체제의 할당된 서비스와의 서비스 결합도가 제1 결합도 임계값 이상인 할당 대상인 서비스가 존재할 경우, 이와 같은 할당 대상인 서비스를 제2 운영체제에 할당할 수 있다.

[0086] 예를 들면, 실시간 서비스 리스트와 비실시간 서비스 리스트를 생성하는 외에, 서비스 관리 모듈은 서비스 분리 평가와 관리도 담당한다. 즉, 하드웨어 리소스 동적 할당 모듈이 프로세서 리소스의 재할당을 수행할 수 있도록, 독립시켜 실시간 운영체제에 넘겨 실행될 수 있는 서비스를 모든 실시간 서비스에서 찾는다. 독립시켜 실시간 운영체제에 넘겨 실행될 수 없는 서비스에 있어서, 이와 비실시간 서비스의 서비스 결합도가 높을 경우, 이를 비실시간 운영체제에 할당할 수 있다.

[0087] 여기서, 어떠한 서비스는 실시간성 요구사항을 갖지만, 이는 시스템에서의 다른 비실시간 서비스와의 인터랙션이 아주 빈번(즉, 서비스 결합도가 높음)하므로, 전반적인 데이터 인터랙션 효율을 향상시키기 위해, 이와 같은 서비스를 비실시간 운영체제에 할당한다. 하지만 다른 일 실시간 서비스는 자체가 상대적으로 독립적이므로, 이를 실시간 운영체제에 구분시키면 된다. 이 과정이 바로 "분리" 동작이다. 서비스를 독립시키는 것을 판단하는 기준은 고유한것이 아니며, 상기 서비스 사이와 연관되는 밀접정도일 수도 있고, 다른 사용자에게 의해 주목받는 지표일 수도 있다.

[0088] 재할당의 전략은 오픈된 것으로서, 한가지 가능한 전략은, 시스템이 처음 실행될 시, 서비스 관리 모듈에서 실시간 운영체제와 비실시간 운영체제에 할당한 서비스 개수의 비율에 따라 프로세서 코어를 할당하고, 후속적인 실행 과정에서 듀얼 시스템에서 각자의 코어 리소스 점유율에 따라 리소스 할당을 조정하는 것이다. 상기 측면에서 말하자면, 재할당 과정은 코어 선점 및 해제 과정과 상호 협력하는 과정이다.

[0089] 선택적인 일 실시형태에서는, 제1 운영체제에서 실행되는 서비스는 하드웨어 인터페이스 신호의 생성 서비스를 포함할 수 있으나, 이에 한정되지 않는다. 본 실시형태에서는 하드웨어 인터페이스 신호의 생성 과정을 제공한다. 상기 과정은 하기와 같은 단계 1, 단계 2 및 단계 3을 포함한다.

[0090] 단계 1에서, 제1 운영체제를 통해 요청 명령을 획득한다. 여기서, 요청 명령은 하드웨어 인터페이스 신호의 생성 명령일 수 있다. 예를 들면, 하드웨어 인터페이스 신호가 PECl 신호일 경우, 요청 명령은 PECl 프로토콜에 기반하는 PECl 요청 명령일 수 있다. 하드웨어 인터페이스 신호는, 예를 들면, HDMI(high definition multimedia interface, 고선명 멀티미디어 인터페이스) 신호, RGMII(reduced gigabit media independent interface, 병렬 버스) 신호, SGMII(serial gigabit media independent interface, 독립적으로 전송되는 직렬 버스) 신호, GPIO(general-purpose input/output, 범용형 입출력 포트) 신호, SPI(serial peripheral interface, 직렬 외부 인터페이스) 신호 등과 같은 다른 프로토콜 유형의 하드웨어 인터페이스 신호일 수도 있다. 이 기초상에서, 요청 명령은, 예를 들면, 하드웨어 인터페이스 신호가 GPIO 신호일 경우, 요청 명령은 GPIO 요청 명령인 것과 같은 다른 프로토콜 유형의 요청 명령일 수도 있다. 본 출원에서는 요청 명령 및 하드웨어 인터페이스 신호의 구체적인 유형에 대해 특별하게 한정하지 않는다.

[0091] 단계 2에서, 요청 명령에 대응되는 복수의 로직비트 정보를 결정한다.

[0092] 단계 2에서는, 제1 운영체제는 요청 명령을 획득한 다음, 요청 명령에 대응되는 복수의 로직비트 정보를 분석하여 획득할 수 있다. 여기서, 복수의 로직비트 정보 사이에는 선후 순서가 존재하고, 제1 운영체제는 요청 명령에 대응되는 복수의 로직비트 정보를 통해 요청 명령에 대응되는 파형 신호(즉, 하드웨어 인터페이스 신호)를 생성할 수 있으므로, 하드웨어 인터페이스 신호를 통해 요청 명령에 포함되는 정보를 다른 기기에 전송할 수 있다.

[0093] 선택적으로, 요청 명령에는 적어도 하나의 필드가 포함되고, 각 필드는 로직비트 0 또는 1로 나타낼수 있다. 이 기초상에서, 각 필드와 로직비트 1 또는 0 사이에 대응되는 변환 관계가 바로 상기 필드에 대응되는 로직비트 정보이다. 요청 명령에 복수의 필드가 대응될 경우, 요청 명령에는 복수의 로직비트 정보가 대응된다. 또한, 각 로직비트는 고레벨 신호와 저레벨 신호의 결합 사용을 통해 나타낼수 있다. 예를 들면, 로직비트 0의 경우, 제1

사전 설정 시간의 고레벨 신호 및 제2 사전 설정 시간의 저레벨 신호를 사용하여 조합하여 나타낼 수 있고, 로직비트 1의 경우, 제2 사전 설정 시간의 고레벨 신호 및 제1 사전 설정 시간의 저레벨 신호를 사용하여 조합하여 나타낼 수 있다. 여기서, 제1 사전 설정 시간과 제2 사전 설정 시간은 서로 다르다. 이 기초상에서, 각 로직비트는 고레벨 신호도 포함하고 저레벨 신호도 포함하므로, 각 로직비트는 실제적으로 한 세그먼트의 파형 신호(고레벨 신호와 저레벨 신호 사이의 변환은 하나의 파형으로 나타남)로 나타낸 것이다. 요청 명령에 복수의 로직비트 정보가 대응, 즉 복수의 로직비트가 대응되므로, 요청 명령에 대응되는 하드웨어 인터페이스 신호는 각 로직비트 정보에 대응되는 파형 신호로 조합되어 획득되는 하나의 파형 신호이다.

[0094] 단계 3에서, 복수의 로직비트 신호 및 타이머에 따라 요청 명령에 대응되는 하드웨어 인터페이스 정보를 생성한다.

[0095] 선택적으로, 단계 3에서의 타이머는 제1 운영체제에서의 타이밍 프로그램일 수 있고, 타이머는 제1 운영체제가 위치하는 칩에서의 레지스터일 수도 있다. 여기서, 타이머는 적어도 타이밍 기능 및 카운트 기능을 제공할 수 있다. 본 출원은 타이머의 타이밍 기능 및 카운트 기능을 적용하고, 복수의 로직비트 정보를 결합하여 요청 명령에 대응되는 하드웨어 인터페이스 신호를 생성한다.

[0096] 유의해야 할 것은, 칩이 BMC 칩이고, 하드웨어 인터페이스 신호가 PECI 신호인 것을 예로 들면, 관련 기술에서는, BMC 칩과 CPU 등 디바이스 사이의 PECI 통신을 실현하기 위해, 관련 기술은 BMC 칩 자체에 PECI 제어기의 하드웨어 로직 디자인이 구비되어야 하므로, BMC 칩의 디자인 비용이 높은 과제를 초래한다. 다시말하면, 관련 기술에서는, BMC 칩에서 PECI 신호를 생성시키기 위해, 사전에 BMC 칩에서 PECI 제어기의 하드웨어 로직 디자인을 실현해야 하지만, 본 출원에서는, BMC 칩에서 PECI 제어기의 하드웨어 로직 디자인을 실현해야 할 필요없이, 제1 운영체제만으로 BMC 칩에서 PECI 신호를 생성시킬 수 있으므로, BMC 칩의 디자인 난이도 및 디자인 비용을 감소시킬 수 있다.

[0097] 상기 단계에 기반하여, 본 실시형태에서는, 제1 운영체제에서 요청 명령에 대응되는 하드웨어 인터페이스 신호를 생성하는 방식을 적용하여, 먼저, 제1 운영체제를 통해 요청 명령을 획득하고, 다음으로 요청 명령에 대응되는 복수의 로직비트 정보를 결정하며, 마지막으로 복수의 로직비트 정보 및 타이머에 따라 요청 명령에 대응되는 하드웨어 인터페이스 신호를 생성한다.

[0098] 상기 내용에서 알 수 있다 같이, 본 실시형태에서는 제1 운영체제를 통해 요청 명령에 대응되는 하드웨어 인터페이스 신호를 생성함으로써, 소프트웨어 방식을 사용하여 하드웨어 인터페이스 신호를 시뮬레이션하여 생성하는 기술적 효과를 실현하며, 나아가 칩에 대해 하드웨어 인터페이스 신호의 하드웨어 로직 디자인을 수행할 필요없이, 소프트웨어 시스템을 사용하여 하드웨어 인터페이스 신호를 생성하는 목적에 달성함으로써, 칩의 디자인 난이도를 낮출 수 있을 뿐만 아니라, 칩의 디자인 비용도 감소시킬 수 있다.

[0099] 선택적으로, 제1 운영체제를 통해 제2 운영체제에 의해 트리거된 제1 요청이 검출될 경우, 요청 데이터를 획득한다. 여기서, 제1 운영체제 및 제2 운영체제는 동일한 프로세서에서 실행되고, 요청 데이터는 제2 운영체제에서 생성되며, 제2 운영체제의 서비스 응답 속도는 제1 운영체제의 서비스 응답 속도보다 늦다. 마지막으로, 제1 운영체제에서 요청 데이터를 분석하여, 요청 명령을 획득한다.

[0100] 선택적으로, 요청 데이터를 획득하기 전에, 제2 운영체제를 통해 요청 데이터를 타깃 메모리(즉, 프러세서에서의 저장 공간)에 저장할 수 있으며, 요청 데이터를 저장 완료한 다음, 제2 운영체제를 통해 제1 요청을 트리거할 수 있다. 여기서, 제1 요청은 제1 운영체제에서 타깃 메모리로부터 요청 데이터를 읽도록 알리기 위한 것이고, 타깃 메모리는 제1 운영체제와 제2 운영체제에서 모두 액세스할 수 있는 메모리이다.

[0101] 선택적인 일 실시예에서는, 제1 운영체제는 하드웨어 인터페이스 신호에 대응되는 응답 데이터를 수신할 수도 있다. 여기서, 응답 데이터의 전송 형태와 하드웨어 인터페이스 신호의 전송 형태는 동일하다. 다음으로, 제1 운영체제는 응답 데이터의 데이터 구조를 제2 데이터 구조로 조정한다.

[0102] 또한, 응답 데이터의 데이터 구조를 제2 데이터 구조로 조정한 다음, 제1 운영체제를 통해 제2 요청을 트리거한다. 여기서, 제2 요청은 제2 운영체제에서 응답 데이터를 읽도록 알리기 위한 것이다.

[0103] 제1 운영체제가 RTOS 시스템이고, 제2 운영체제가 Linux 시스템이며, 하드웨어 인터페이스 신호가 PECI 신호인 것을 예로 든다. 명령 요청 과정에 있어서, 먼저 Linux 시스템에서 PECI 서비스에 관한 상위 계층 응용(예를 들면, 고장 진단, CPU 온도 획득 등)은 수요에 따라 능동적으로 PECI 요청 명령을 발신하며, 상기 요청 명령은 기본적인 Ping(Packet Internet Groper) 명령을 포함하지만, 이에 한정되지 않고, CPU 온도 명령 및 MRS 레지스터의 정보를 읽는 명령 등을 획득하며, 서로 다른 PECI 요청 명령의 코드의 실행은 대응되는 인터페이스 함수로

완성한다.

- [0104] 선택적으로, Linux 시스템은 PECI 프로토콜 스펙에 따라 각 요청 명령의 타깃 주소, 읽기/쓰기 길이, 명령 코드, para 파라미터 등 요청 데이터를 타깃 메모리에 입력하고, 요청된 데이터를 타깃 메모리에 전부 입력한 다음, Linux 시스템에서 제1 요청을 생성하며, RTOS 시스템에 알린다. 여기서, 제1 요청은 SGI 중단 요청 (software generated interrupt, 프로세서 코어 사이의 통신 중단 요청)일 수 있다.
- [0105] 유의해야 할 것은, 제2 운영체제를 통해 요청 데이터를 타깃 메모리에 저장하는 과정에서, 제2 운영체제는 요청 데이터를 제1 데이터 구조의 형태에 따라 타깃 메모리에 저장한다. 여기서, 제1 데이터 구조는 적어도 기기 주소, 쓰기 길이, 읽기 길이, 명령 코드 및 요청 파라미터를 포함하며, 기기 주소는 타깃 디바이스의 주소를 표시하기 위한 것이고, 타깃 디바이스는 하드웨어 인터페이스 신호에 근거하여 응답 데이터를 생성하는 디바이스이며, 명령 코드는 서로 다른 요청 명령을 구분하기 위한 것이고, 쓰기 길이는 명령 코드로부터 시작하여 요청 데이터 종료까지의 바이트 개수를 표시하기 위한 것이며, 읽기 길이는 요청 데이터에 완성 코드 및 읽기 데이터를 포함하는 바이트 개수가 포함되는 것을 표시하기 위한 것이고, 요청 파라미터는 요청 명령의 파라미터를 표시하기 위한 것이다.
- [0106] 명령 응답 과정에 있어서, RTOS 시스템에서 PECI 버스로부터 전송된 응답 데이터를 수신한 다음, 데이터 분석을 완성하여, 응답 데이터의 신호 형태를 하드웨어 인터페이스 신호의 형태로부터 소프트웨어 신호의 형태로 변환시킨다. 예를 들면, 하드웨어 인터페이스 신호에서의 고레벨 신호와 저레벨 신호 사이의 파형 변화를 식별하여, 대응되는 로직비트 정보를 획득하고, 로직비트 정보에 기반하여 소프트웨어 정보 데이터를 획득한다. 분석 후의 응답 데이터는 명령 파라미터 구조화 모듈을 통해 조정되어 타깃 메모리에 입력된다. 분석 후의 응답 데이터가 전부 입력 완료된 후, RTOS 시스템에서 제2 요청을 트리거하여 Linux 시스템에 알린다. Linux 시스템에서 제2 요청을 검출하고, 능동적으로 타깃 메모리에 저장된 분석 후의 응답 데이터를 읽으며, 데이터는 처리된 후 상위 계층 응용에 반환된다. 여기서, 제2 요청은 SGI 중단 요청일 수 있다.
- [0107] 타깃 메모리는 공용 메모리일 수 있는 외에, 예를 들면, 랜덤 액세스 메모리(Random Access Memory, RAM으로 약칭함), 플래시 메모리(Flash) 등등과 같은 다른 메모리일 수도 있다.
- [0108] 선택적인 일 실시예에서는, 로직비트 정보 및 타이머에 따라 요청 명령에 대응되는 하드웨어 인터페이스 신호를 생성한 다음, 제1 운영체제는 하드웨어 인터페이스 신호의 전압을 변환하여, 타깃 하드웨어 인터페이스 신호를 획득할 수 있다.
- [0109] 선택적으로, 제1 운영체제는 하드웨어 인터페이스 신호를 전압 변환 디바이스에 입력하여, 전압 변환 디바이스에서 출력되는 타깃 하드웨어 인터페이스 신호를 획득할 수 있다. 상기 전압 변환 디바이스는 CPLD일 수 있으며, CPLD는 타깃 디바이스와 서로 연결될 수 있다. 여기서, 타깃 디바이스는 서버에서의 CPU일 수 있다. 유의해야 할 것은, 상기 서비스는 PECI 인터페이스를 대체하여 PECI 신호를 생성하는데 응용되는 외에, 다른 하드웨어 인터페이스에 응용될 수도 있다.
- [0110] 상기 내용에서 알 수 있다 싶이, 연합 임베디드 시스템의 제1 운영체제와 제2 운영체제는, 코어 간 중단과 메모리 공유의 방식을 통해 임베디드 시스템에서 데이터의 인터랙션을 실현하고, RTOS 시스템에서 요청 명령을 생성하는 파형 발생 기능 모듈은, 소프트웨어 시뮬레이션 방식을 통해 임베디드 시스템과 외부 디바이스 사이에서 하드웨어 인터페이스 신호의 통신을 수행한다. 또한, RTOS 시스템의 높은 실시간성 특성을 충분히 이용하여, 요청 명령 파형을 시뮬레이션할 시 타이밍 시퀀스의 정확성을 보장하고, 원활하고 고효율적인 특성을 갖도록 하며, 칩 디자인의 난이도를 현저하게 낮출 수 있다. 소프트웨어 시뮬레이션을 적용하여 하드웨어 인터페이스 신호를 생성하므로, 임베디드 시스템에서 통신 기능과 다른 서비스 기능 간의 최적화 디자인을 위해 더 많은 가능성을 제공하고, 동시에 칩에서 하드웨어 인터페이스 신호 통신에 특별히 사용되는 제어기가 생략되어, 칩의 디자인 비용 및 제조 비용을 감소시킬 수 있다.
- [0111] 선택적인 일 실시예에서는, 제1 운영체제에서 실행되는 서비스는 직렬포트 전환 서비스를 포함할 수 있지만, 이에 한정되지 않는다. 본 실시예에서는 직렬포트의 전환 과정을 제공한다. 상기 과정은 하기와 같은 단계 1 및 단계 2를 포함한다.
- [0112] 단계 1에서, 제2 운영체제에서 직렬포트 전환 명령어를 수신한 것이 검출될 경우, 제2 운영체제를 통해 직렬포트 전환 명령어를 제1 운영체제에 발신한다.
- [0113] 선택적으로, 사용자가 직렬포트 전환을 발신할 경우, 제2 운영체제를 통해 사용자에게 의해 발신된 직렬포트 전환 명령어의 수신 여부를 검출할 수 있다. 설명해야 할 것은, 직렬포트 전환 명령어에는 전환될 타깃 직렬포트의

정보가 포함되어야 한다. 예를 들면, 직렬포트 전환 명령어는 전환될 타깃 직렬포트의 직렬포트 번호를 포함한다.

- [0114] 선택적인 일 실시예에서는, 직렬포트 전환 명령의 포맷은 <switch_command_app-n number-t sleep_time>일 수 있다. switch_command_app은 전환 명령어 프로그램을 표시하고, -n은 전환되는 타깃 직렬포트 번호를 나타내며, number의 값은 1, 2, 3일 수 있고, -t는 명령어 발신 후로부터 얼마동안 휴면 후 전환 동작을 실행하는지를 나타내며, sleep_time의 단위는 초이다.
- [0115] 설명해야 할 것은, 후속적으로 직렬포트 전환을 수행할 시 직렬포트 번호를 통해 타깃 직렬포트에 대한 전환을 실현할 수 있도록, 직렬포트 전환을 실현할 시 현재 직렬포트 전환을 수행할 수 있는 직렬포트를 위해 번호를 매길 수 있다.
- [0116] 선택적인 일 실시예에서는, 현재 직렬포트 전환을 수행할 수 있는 직렬포트는 BMC Linux 시스템 직렬포트, 서버 BIOS(Basic Input Output System, 기본 입출력 시스템) 직렬포트 및 SMART NIC(network interface controller, 지능 네트워크 인터페이스 제어기) 직렬포트를 포함한다. 대응되게, 1로 BMC Linux 시스템 직렬포트를 나타낼 수 있고, 2로 서버 BIOS 직렬포트를 나타낼 수 있으며, 3으로 SMART NIC 직렬포트를 나타낼 수 있다.
- [0117] 단계 2에서, 제1 운영체제를 통해 직렬포트 전환 명령어에 근거하여 직렬포트 전환을 실행한다.
- [0118] 선택적으로, 제2 운영체제에서 직렬포트 전환 명령어를 수신한 것이 검출될 경우, 제2 운영체제는 즉시 직렬포트 전환 명령어를 제1 운영체제에 발신한다. 설명해야 할 것은, 제1 운영체제와 제2 운영체제를 2개의 프로세서 코어에서 각각 실행한 다음, 제1 운영체제와 제2 운영체제 사이에 코어 간 통신을 적용한다. 이렇게 되면, 신호 전달의 신뢰성을 향상시키는데 도움된다.
- [0119] 설명해야 할 것은, 제1 운영체제에서 명령어에 대한 응답 속도는 제2 운영체제에서 명령어에 대한 응답 속도보다 훨씬 빠르다. 이렇게 되면 제1 운영체제를 통해 직렬포트 전환 명령어에 신속하게 응답할 수 있으며, 아주 짧은 시간 내에 전환 동작을 완성할 수 있다.
- [0120] 상술한 바와 같이, 동일한 프로세서에서 실행되는 제1 운영체제 및 제2 운영체제를 통해 CPLD 또는 FPGA를 대체하여 직렬포트 전환 소프트웨어 기능을 실현하고, 제2 운영체제에서 직렬포트 전환 명령어를 수신할 경우, 제2 운영체제를 통해 직렬포트 전환 명령어를 제1 운영체제에 전달하며, 제1 운영체제에서 직렬포트 전환 명령어에 따라 직렬포트 전환을 실현하여, 관련 기술에서 CPLD 또는 FPGA를 통해 각 직렬포트를 연결시킨 다음, CPLD 또는 FPGA에서의 스위치 구조를 이용하여 직렬포트 전환을 실현해야 하는 것을 방지하고, 하드웨어 비용을 감소하며, 또한 제1 운영체제에서 직렬포트 전환 명령어를 수신한 후, 신속하게 짧은 시간 내에 직렬포트 전환을 완성할 수 있다. 따라서, 본 방안에서 제시되는 해결수단은 직렬포트 전환 비용을 효과적으로 감소시킬 수도 있고, 직렬포트 전환의 효율을 효과적으로 향상시킬 수도 있다.
- [0121] 제2 운영체제에서 직렬포트 전환을 실현할 수 있게 하기 위해, 본 실시형태에서 제공되는 직렬포트 전환 과정에서, 직렬포트 전환 명령어는 적어도 타깃 직렬포트의 직렬포트 번호를 포함하며, 제1 운영체제를 통해 직렬포트 전환 명령어에 근거하여 직렬포트 전환을 실행하는 단계 이전에, 제1 운영체제를 통해 타깃 메모리에서 직렬포트 전환 명령어의 분석 규칙을 획득하는 단계; 분석 규칙에 근거하여 직렬포트 전환 명령어에서의 타깃 직렬포트의 직렬포트 번호를 분석하여, 직렬포트 번호에 대응되는 기기를 결정하되, 여거서, 타깃 직렬포트는 기기의 직렬포트이고, 타깃 직렬포트는 칩에 연결되는 것인 단계; 를 포함한다.
- [0122] 제1 운영체제를 통해 직렬포트 전환 명령어에 근거하여 직렬포트 전환을 실행하는 단계는, 제1 운영체제를 통해 기기의 직렬포트 주소를 결정하는 단계; 직렬포트 주소에 근거하여 타깃 직렬포트를 칩의 타깃 출력 인터페이스에 매핑하는 단계; 를 포함한다.
- [0123] 제1 운영체제에서 직렬포트 전환을 실현할 수 있도록 하기 위해, 제1 운영체제에서는 직렬포트 전환 명령어를 분석할 수 있으며, 나아가 타깃 직렬포트에 대응되는 기기를 획득할 수 있다.
- [0124] 선택적인 일 실시예에서, 칩 또는 서버 메인보드가 서로 다른 것에 근거하여 직렬포트 전환 명령어에 대한 분석 규칙을 사용자 지정할 수 있으며, 분석 규칙을 타깃 메모리에 저장할 수 있다. 타깃 메모리는 전기적 소거 및 프로그래밍 가능 리드 온리 메모리(eeprom), 비휘발성 메모리(flash) 등 저장 매체일 수 있다. 설명해야 할 것은, 타깃 메모리는 칩에 배치될 수도 있고, 칩에 배치되지 않을 수도 있다. 타깃 메모리를 통해 분석 규칙을 저장하여, 데이터의 보안성을 향상시킬 수 있고, 칩 또는 서버 메인보드의가 서로 다른 것에 근거하여 분석 규칙을 사용자 지정할 수 있어, 프로그래밍 가능성 및 확장 가능성이 우수해지도록 할 수 있다.

- [0125] 제1 운영체제에서 직렬포트 전환 명령어를 수신한 후, 타깃 메모리로부터 직렬포트 전환 명령어의 분석 규칙을 읽고, 다음 분석 규칙을 이용하여 직렬포트 전환 명령어에서의 타깃 직렬포트의 직렬포트 번호를 분석하여, 상기 직렬포트에 대응되는 기기를 획득한다.
- [0126] 직렬포트 번호에 대응되는 기기를 획득한 후, 제1 운영체제에서는 기기의 직렬포트 주소를 통해 타깃 직렬포트를 칩의 타깃 출력 인터페이스에 매핑할 수 있다. 기기의 직렬포트 주소를 타깃 출력 인터페이스에 매핑한 후, 타깃 출력 인터페이스를 통해 기기에 대한 액세스를 실현할 수 있다. 설명해야 할 것은, 직렬포트 전환 명령어 및 분석 규칙은 사용되는 칩의 모델 및 제1 운영체제와 제2 운영체제의 유형에 따라 설정될 수 있다.
- [0127] 본 출원의 실시예에 의해 제공되는 직렬포트 전환 방법에서는, 칩은 직렬 데이터 버스를 포함하고, 제1 운영체제를 통해 기기의 직렬포트 주소를 결정하는 단계 이후, 이 방법은 직렬 데이터 버스의 직렬포트와 연결되는 복수의 기기를 결정하는 단계; 직렬 데이터 버스를 통해 기기 각각의 직렬포트를 칩의 메모리에 매핑하여, 기기 각각의 직렬포트 주소를 획득하는 단계; 를 더 포함한다.
- [0128] 선택적으로, 상기 칩에는 직렬 데이터 버스가 더 포함되고. 현재 복수의 기기의 직렬포트의 TX와 RX는 직렬 데이터 버스와 서로 연결된다. 예를 들면, 현재의 직렬포트는 BMC Linux 시스템 직렬포트(UART1), 서버 BIOS 직렬포트(UART2) 및 SMART NIC 직렬포트(UART3)를 포함한다. UART(Universal Asynchronous Receiver/Transmitter)는 범용 비동기화 송수신기이다. 직렬포트 데이터 버스는 UART1, UART2 및 UART3의 서로 다른 직렬포트의 TX와 RX 데이터를 BMC 메모리의 서로 다른 주소 공간에 매핑한다. 즉, 상술한 직렬 데이터 버스를 통해 기기 각각의 직렬포트를 칩의 메모리에 매핑한다. 예를 들면, UART1 TX와 RX buffer는 직렬포트 UART1의 직렬포트 주소이고, UART2 TX와 RX buffer는 직렬포트 UART2의 직렬포트 주소이며, UART3 TX와 RX buffer는 직렬포트 UART3의 직렬포트 주소이다.
- [0129] 사용자가 직렬포트 전환 명령어를 발신할 경우, 제1 운영체제(RTOS)에서 UART에 의해 매핑된 서로 다른 3세그먼트의 메모리를 선택(삼지선다형)하고, 그 중 1세그먼트의 메모리 데이터를 사용자에게 인터랙션하여, CPLD 하드웨어 직렬포트 전환 회선을 시뮬레이션하는 목적에 달성한다. 설명해야 할 것은, 서로 다른 기기의 직렬포트를 구분할 수 없을 경우, 개발자는 유지보수 시 어느 기기의 직렬포트에 문제가 존재하는지 정확하게 포지셔닝할 수 없으므로, 직렬포트 전환을 통해 예외 문제에 대한 포지셔닝을 실현해야 한다.
- [0130] 본 실시형태에서는, 직렬포트 주소에 근거하여 타깃 직렬포트를 칩의 타깃 출력 인터페이스에 매핑하는 단계 이후, 타깃 출력 인터페이스가 타깃 지능 네트워크 카드와 연결될 경우, 상기 단계는, 지능 네트워크 카드를 통해 타깃 직렬포트에 대한 액세스 요청을 수신하였는지 여부를 검출하는 단계; 타깃 직렬포트에 대한 액세스 요청을 수신할 경우, 지능 네트워크 카드를 통해 액세스 요청을 타깃 직렬포트에 전달하는 단계; 를 포함한다.
- [0131] 선택적으로, 칩의 타깃 출력 인터페이스에서 타깃 지능 네트워크 카드와 연결한 다음, 지능 네트워크 카드를 통해 사용자가 타깃 직렬포트에 대한 액세스 요청을 수신하였는지 여부를 검출할 수도 있으며, 타깃 직렬포트에 대한 액세스 요청을 수신할 경우, 직접 타깃 지능 네트워크 카드를 통해 기기의 직렬포트에 대한 액세스를 실현할 수 있고, SOL(Serial over LAN, 데이터 패킷 포맷 및 프로토콜의 스펙) 기능을 실현할 수 있다. 상기 단계를 통해, 기기의 직렬포트에 대한 액세스 효율을 향상시킨다.
- [0132] 선택적인 일 실시예에서는, 직렬포트 주소에 근거하여 타깃 직렬포트를 칩의 타깃 출력 인터페이스에 매핑하는 단계 이후, 제1 운영체제를 통해 직렬포트 전환 명령어의 실행 결과를 획득하되, 실행 결과는 전환 성공 및 전환 실패 중의 하나인 것인 단계; 제1 운영체제를 통해 실행 결과를 제2 운영체제에 발신하는 단계; 를 더 포함한다.
- [0133] 제2 운영체제를 통해 직렬포트 전환 명령어의 실행 결과를 수신한다. 여기서, 실행 결과는 제1 운영체제에서 제2 운영체제에 발신하고, 실행 결과는 직렬포트 전환 성공 및 직렬포트 전환 실패 중의 하나이다. 제1 운영체제에서 직렬포트 전환을 완료한 후, 직렬포트 전환 명령어의 실행 결과를 획득한 다음, 직렬포트 전환 명령어의 실행 결과를 제2 운영체제에 피드백하며, 제2 운영체제에 직렬포트 전환의 성공 또는 실패를 알린다.
- [0134] 직렬포트 전환의 성공율을 향상시키기 위해, 본 실시형태에서는, 제2 운영체제를 통해 직렬포트 전환 명령어의 실행 결과를 수신하는 단계 이후, 실행 결과가 실행 실패일 경우, 실행 결과가 성공일 때까지, 제2 운영체제를 통해 직렬포트 전환 명령어를 제1 운영체제에 발신하는 단계를 중복으로 실행하거나, 또는 직렬포트 전환의 회수가 사전 설정 회수를 초과하여 실행하는 단계; 직렬포트 전환의 회수가 사전 설정 회수를 초과하여 실행할 경우, 제2 운영체제를 통해 알림 신호를 트리거하되, 알림 신호는 직렬포트 전환 실패를 알리기 위한 것인 단계; 를 더 포함한다.

- [0135] 직렬포트 전환 명령어의 실행 결과가 실행 실패일 경우, 실행 결과가 성공일 때까지, 제2 운영체제를 통해 직렬포트 전환 명령어를 제1 운영체제에 발신하는 단계를 중복으로 실행하거나, 또는 직렬포트 전환의 회수가 사전 설정 회수를 초과하여 실행하며, 사전 설정 회수는 3회로 설정될 수 있다. 직렬포트 전환의 회수가 사전 설정 회수를 초과하여 실행할 경우, 대응되는 제2 운영체제에서 알림 신호를 트리거하는 것을 통해, 직렬포트 전환 실패를 알려, 이러한 상황에 대해 적시로 처리하도록 한다.
- [0136] 제1 운영체제에서 직렬포트 전환 명령어를 수신한 것이 검출되기 전에, 제2 운영체제가 시작된 후, 제2 프로세서 코어에서 제1 중단을 트리거하여, 제1 신호를 제1 운영체제에 발신하는 단계; 제1 운영체제를 통해 제1 신호에 근거하여 칩에서의 복수의 직렬포트의 실행 상태를 검출하여, 검출 결과를 획득하는 단계; 제1 프로세서 코어를 통해 제2 중단을 트리거하여, 검출 결과를 제2 신호를 통해 제2 운영체제에 발신하는 단계; 제2 운영체제를 통해 검출 결과를 수신하여, 칩에서의 정상적으로 실행되는 직렬포트의 개수를 결정하는 단계; 를 더 포함한다.
- [0137] 제2 프로세서 코어에서 제1 중단을 트리거하여, 제1 신호를 제1 운영체제에 발신한 다음, 제1 운영체제에서 제1 신호를 수신하였는지 여부를 검출한다. 제1 운영체제에서 제1 신호를 수신할 경우, 제1 운영체제를 통해 칩에서 복수의 직렬포트의 실행 상태를 검출하여, 검출 결과를 획득한다.
- [0138] 제2 운영체제가 시작된 후, 제2 프로세서 코어에서 제1 중단(IPI 중단, IPI, inter-processor interrupts, 프로세서 간 중단)을 트리거하여, 제1 신호를 제1 운영체제에 발신하고, 제1 운영체제에서는 제1 신호를 통해 제2 운영체제가 정상적으로 시작되었음을 알 수 있어, 제2 운영체제와 정상적인 인터랙션을 수행할 수 있으며, 제1 운영체제에서는 제1 신호에 따라 칩에서의 복수의 직렬포트의 실행 상태를 검출하여, 모든 직렬포트가 정상적으로 실행되고 있는지 여부를 결정할 수 있다.
- [0139] 제1 운영체제에서 검출 결과를 검출하여 획득한 후, 제1 프로세서 코어에서 제2 중단을 트리거하여 검출 결과를 제2 신호를 통해 제2 운영체제에 발신하고, 제2 운영체제에서는 검출 결과를 통해 전환될 수 있는 직렬포트 개수(즉, 상기 정상적으로 실행되는 직렬포트의 개수)를 결정하여, 후속적으로 이러한 직렬포트에 대해 직렬포트 전환을 수행할 수 있도록 한다. 동시에 제1 운영체제에서 직렬포트 전환을 보다 더 신속하게 실현할 수 있도록 하기 위해, 제1 운영체제의 검출이 완료된 후, 제1 운영체제는 블로킹되기 시작하고, 제2 운영체제에서 발신되는 직렬포트 전환 명령어에 대한 수신을 대기한다.
- [0140] 선택적인 일 실시예에서는, 제1 운영체제가 RTOS이고, 제2 운영체제가 Linux일 경우, 제1 운영체제는 CPU0에서 실행되고, 제2 운영체제는 CPU1에서 실행된다. 직렬포트 전환 전의 준비 단계는, CPU1에서의 Linux 시스템이 특정된 단계에서 시작될 때, CPU1에서 하나의 IPI 중단을 트리거하고, CPU0에서의 RTOS 시스템에 Linux이 정상적으로 시작되었음을 알리며, CPU1에서의 Linux와 정상적인 인터랙션을 수행할 수 있으며, RTOS 시스템에서 CPU1로부터의 IPI 중단을 수신한 후, 직렬포트 전환 제어기 프로그램을 시작하여, UART1, UART2, UART3의 정상 여부를 체크한 다음, CPU0에서 IPI 중단을 재트리거하고, CPU1에서의 Linux 시스템에 RTOS 시스템이 정상적으로 시작 완료되었음을 알리며, 동시에 보고되는 정보는 CPU0에서의 RTOS 시스템이 갖는 전환할 수 있는 직렬포트의 개수를 포함하고, 다음 CPU0에서의 RTOS 시스템은 블로킹되기 시작하고 CPU1에서의 운영체제에서 발신되는 전환 명령어에 대한 수신을 대기하는 단계를 포함한다.
- [0141] 제2 운영체제의 실행에 예외가 존재할 경우, 서버 단말기를 통해 직렬포트 전환 명령어를 제1 운영체제에 발신하고, 제1 운영체제를 통해 직렬포트 전환 명령어에 근거하여 직렬포트 전환을 실행한다.
- [0142] 제2 운영체제에서 실행되는 기능이 많고, 담당하는 서비스량이 크므로, 실행 예외 또는 재부팅해야 할 경우가 존재할 수 있다. 제2 운영체제의 실행에 예외가 존재할 경우, 직접 서버 단말기를 통해 직렬포트 전환 명령어를 제1 운영체제에 발신하여, 제1 운영체제에서 직렬포트 전환을 정상적으로 실행하도록 보장할 수 있다. 설명해야 할 것은, 서버 단말기는 칩이 위치한 서버에서의 단말기일 수 있다.
- [0143] 상기 단계를 통해, 제1 운영체제에서 제2 운영체제에 의존하지 않고 직렬포트의 전환을 실현하도록 보장하고, 제1 운영체제에서 직렬포트 전환을 실행하는 독립성을 향상시킨다.
- [0144] 상술한 바와 같이, 본 실시형태에 의해 제공되는 직렬포트 전환 과정에서, 동일한 프로세서에서 실행되는 제1 운영체제 및 제2 운영체제를 통해 CPLD 또는 FPGA를 대체하여 직렬포트 전환 소프트웨어 기능을 실현하고, 제2 운영체제에서 직렬포트 전환 명령어를 수신할 경우, 제2 운영체제를 통해 직렬포트 전환 명령어를 제1 운영체제에 전달하며, 제1 운영체제에서 직렬포트 전환 명령어에 따라 직렬포트 전환을 실현하여, 하드웨어의 방식을 적용하여 직렬포트 전환을 실현하는 것을 방지하고, 하드웨어 비용을 감소하며, 또한 제1 운영체제에서 직렬포트

전환 명령어를 수신한 후, 신속하게 짧은 시간 내에 직렬포트 전환을 완성할 수 있다. 따라서, 상기 과정은 직렬포트 전환 비용을 효과적으로 감소시킬 수도 있고, 직렬포트 전환의 효율을 효과적으로 향상시킬 수도 있다.

- [0145] 본 실시예에서, 서비스 결합도를 통해 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 통해, 서비스 결합도가 높은 복수의 서비스에 대해 서비스 처리를 수행하는 정확성을 보장할 수 있다.
- [0146] 예시적인 일 실시예에서는, 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당하는 단계는 하기와 같은 단계 S61을 포함한다.
- [0147] S61에서, 한세트의 할당 대상인 서비스에 민감한 정보가 포함되는 할당 대상인 서비스를 타깃 운영체제에 할당한다. 여기서, 타깃 운영체제는 제1 운영체제 및 제2 운영체제에서 사용 객체와 인터랙션하는 빈도가 낮은 운영체제이다.
- [0148] 본 실시예에서는, 민감한 데이터(예를 들면, 비밀번호 등 민감한 정보)를 포함하는 할당 대상인 서비스(이는, 예를 들면, 사용자에게 노출되지 않는 것을 원하는 서비스와 같은 중요한 민감형 서비스일 수 있음)의 경우, 이는 타깃 운영체제에 할당될 수 있으며, 타깃 운영체제를 통해 민감한 정보를 포함하는 할당 대상인 서비스에 대해 하드 코어 레벨의 보안 방호 격리를 수행할 수 있다. 여기서, 타깃 운영체제는 제1 운영체제 및 제2 운영체제에서 사용 객체와 인터랙션하는 빈도가 낮은 운영체제이거나, 또는, 예를 들면, 제1 운영체제와 같이 응답 속도가 빠른 운영체제이다.
- [0149] 예를 들면, 상기 서비스 처리 모듈은 진일보로 시스템의 서비스에 대해 하드 코어 레벨의 보안 방호 격리를 수행하는 것을 담당한다. 즉, 중요한 민감형(사용자에게 노출되지 않는 것을 원함)의 서비스를 실시간 서비스로 구분하고, 최종적으로 이러한 서비스가 비실시간 운영체제에서 실시간 운영체제로의 언로딩을 실현할 수 있어, 보안 방호의 효과를 일으킬 수 있다. 여기서, 상기 서비스 처리 모듈에서 구분된 서로 다른 서비스는 소프트웨어에서 실현 시 구조체의 형태를 적용하여 구성될 수 있다. 이기중 운영체제 간의 보안 공간을 디자인하여, 민감형 서비스를 비실시간 운영체제에서 실시간 운영체제로 언로딩하여, 하드 코어 레벨의 보안 방호 목적에 달성한다. 여기서, 민감형 서비스는, 예를 들면, 사용자 비밀번호, 신분 정보 등 사용자 개인 프라이버시에 관한 서비스와 같은 보안과 관련되는 서비스를 가리킨다.
- [0150] 여기서, 하드 코어 레벨은 서비스가 프로세서 코어 분야에서 격리되는 것을 가리킨다. 즉, 민감형 서비스를 실시간 운영체제(실시간 운영체제가 있는 코어는 비실시간 운영체제와 다르므로, 코어 분야의 격리에 속함)에 할당하고, 실시간 운영체제를 비실시간 운영체제와 비교할 경우, 사용자와 인터랙션하는 빈도와 정도가 상대적으로 약하므로, 사용하는 객체로서의 사용자는 상기 실시간 운영체제에서 실행되는 서비스에 생성된 민감 데이터를 탐지하기 어렵다. 상위 계층의 응용의 경우, 사용자의 신분 인증 관리, 보안 암호화 등 서비스는 상기 중요한 민감형 서비스에 속하고, 서비스 관리 모듈을 통해 상기 서비스를 실시간 서비스로 강제로 구분하며, 후속적으로 하드웨어 리소스 동적 할당을 수행할 시 상기 서비스가 실시간 운영체제에서의 실행을 실현하여, 보안 격리 효과를 일으킬 수 있다.
- [0151] 본 실시예에서, 민감한 정보를 포함하는 할당 대상인 서비스를 사용자와의 인터랙션 빈도가 낮은 운영체제에 할당하는 단계를 통해, 시스템의 서비스에 대해 하드 코어 레벨의 보안 방호 격리를 수행할 수 있고, 서비스 실행의 보안성을 향상시킬 수 있다.
- [0152] 예시적인 일 실시예에서는, 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 단계는 하기와 같은 단계 S71을 포함한다.
- [0153] S71에서, 한세트의 할당 대상인 서비스의 할당 결과에 따라, 제1 운영체제의 처리 리소스의 리소스 이용 상황 및 제2 운영체제의 처리 리소스의 리소스 이용 상황을 결합하여, 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성한다.
- [0154] 본 실시예에서는, 한세트의 할당 대상인 서비스의 할당 결과는 할당 대상인 서비스와 운영체제 사이의 대응관계를 지시하기 위한 것이다. 하나의 운영체제에 할당되는 실행 대상인 서비스는 통상적으로 상기 운영체제의 처리 리소스를 사용하여 실행되며, 어느 한 운영체제에 할당되는 서비스량이 지나치게 많고, 또한 현재 할당되지 않은 처리리소스가 존재할 경우, 어느 한 운영체제에 할당되는 할당 대상인 서비스에 할당되지 않은 처리 리소스를 할당할 수 있다. 따라서, 한세트의 할당 대상인 서비스의 할당 결과에 따라, 제1 운영체제의 처리 리소스의 리소스 이용 상황 및 제2 운영체제의 처리 리소스의 리소스 이용 상황을 결합하여, 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성하여, 할당 대상인 서비스 각각에 할당되는 처리 리

소스를 지시할 수 있다.

- [0155] 여기서, 할당 대상인 서비스 각각은 단지 어느 한 프로세서 코어와 매핑 관계를 갖고, 동일한 프로세서 코어는 복수의 할당 대상인 서비스와 매핑 관계를 가질 수 있으며, 서로 다른 서비스는 동일한 프로세서 코어를 점유하는 서로 다른 타임 슬라이스를 통해 동일한 프로세서 코어와 매핑 관계를 갖는다. 동일한 시간에서, 동일한 프로세서 코어는 단지 하나의 서비스에 의해 점유된다. 즉, 하나의 서비스를 실행하는데에만 사용된다. 하나의 운영체제에 할당되는 서로 다른 서비스는 할당 시간, 서비스 응답 속도 요구사항 또는 다른 방식에 따라 동일한 프로세서 리소스를 점유하는 타임 슬라이스를 결정할 수 있다.
- [0156] 예를 들면, 리소스 동적 할당 모듈은 서비스 관리 모듈의 출력 결과에 따라, 프로세서 리소스를 동적 조정하여, 서로 다른 서비스와 실제 하드웨어 리소스의 리소스 매핑 테이블을 형성하고, 서로 다른 하드웨어 리소스가 이 기종 운영체제에서의 배치 구조를 최적화하여, 전반 시스템 하드웨어 리소스의 이용율을 향상시키는 목적에 달성한다. 상기 리소스 동적 할당 과정은 제2 운영체제에서의 소프트웨어에 의해 관리 구성된다.
- [0157] 8코어 프로세서(코어1 내지 코어8)를 예로 들면, 제1 운영체제에 스케줄링된 프로세서 코어는 코어1을 포함하고, 제2 운영체제에 스케줄링된 프로세서 코어는 코어2, 코어3 및 코어4를 포함하며, 할당 대상인 서비스는 6개 있고, 실시간성 서비스는 서비스1 및 서비스2이며, 비실시간성 서비스는 서비스3 내지 서비스6이고, 6개의 서비스에 대응되는 프로세서 코어를 할당하며, 서비스1에 코어1을 할당하고, 서비스2에 코어5를 할당하며, 서비스3에 코어2를 할당하고, 서비스4에 코어3을 할당하며, 서비스5에 코어4를 할당하고, 서비스6에 코어6을 할당한다.
- [0158] 본 실시예에서, 서비스와 운영체제의 대응관계에 기반하여, 서로 다른 운영체제의 처리 리소스의 사용 상황을 결합하여 처리 리소스의 동적 할당을 수행하는 단계를 통해, 처리 리소스 할당의 합리성을 보장할 수 있다.
- [0159] 예시적인 일 실시예에서는, 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 단계는 하기와 같은 단계 S81을 포함한다.
- [0160] S81에서, 리소스 할당 결과에 따라 프로세서의 처리 리소스에서의 할당되지 않은 처리 리소스에는 대응되는 할당 대상인 서비스가 존재하는 것으로 결정할 경우, 할당되지 않은 처리 리소스를 할당되지 않은 처리 리소스와 대응되는 할당 대상인 서비스에 의해 할당되는 운영체제에 할당한다.
- [0161] 처리 리소스를 할당할 시, 프로세서의 처리 리소스에서의 할당되지 않은 처리 리소스에는 대응되는 할당 대상인 서비스가 존재한다. 즉, 할당 대상인 서비스에 할당되지 않은 처리 리소스를 할당하고, 할당되지 않은 처리 리소스를 할당되지 않은 처리 리소스와 대응되는 할당 대상인 서비스에 의해 할당되는 운영체제에 할당한다.
- [0162] 선택적으로, 리소스 적응형 스케줄링 모듈은 하드웨어 리소스 동적 할당의 결과에 따라, 프로세서의 처리 리소스에 대한 실제적인 스케줄링 동작을 완성할 수 있다. 리소스 적응형 스케줄링 모듈은, 예를 들면, 도 4에 도시된 코어세트1의 M개 코어와 같이, 일부분의 프로세서 코어를 스케줄링하여 제1 운영체제에 할당되는 서비스를 실행하고, 예를 들면, 도 4에 도시된 코어세트2의 N개 코어와 같이, 나머지 프로세서 코어를 스케줄링하여 제2 운영체제에 할당되는 서비스를 실행한다.
- [0163] 상기 8코어 프로세서를 예로 들면, 서비스 할당 결과 및 리소스 할당 결과에 따라, 할당되지 않은 코어4를 제1 운영체제에 할당할 수 있고, 할당되지 않은 코어5 및 코어6을 Linux 시스템에 할당할 수 있다. 전반 스케줄링 과정은 제2 운영체제에서 주도할 수 있다.
- [0164] 본 실시예에서, 리소스 할당 결과에 기반하여 할당되지 않은 프로세서 리소스를 대응되는 운영체제에 스케줄링하는 단계를 통해, 프로세서 리소스의 이용율을 향상시킬 수 있다.
- [0165] 예시적인 일 실시예에서는, 상기 방법은 단계 S91을 더 포함한다.
- [0166] S91에서, 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0167] 본 실시예에서는, 할당 대상인 서비스에 기반하여 프로세서 리소스의 동적 할당을 수행하는 외에, 각 운영체제가 실행되는 과정에서, 운영체제의 처리 리소스의 사용 상태(로드 상황)에 기반하여, 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행할 수 있다. 여기서, 처리 리소스의 동적 할당은 제1 운영체제 및 제2 운영체제 이외의 다른 제어 로직 모듈에 의해 실행되는 리소스 스케줄링 방식일 수 있고, 처리 리소스의 선점과 해제는 제1 운영체제 및 제2 운영체제에 의해 능동적으로 발신되는 리소스 스케줄링 방식일 수 있다.
- [0168] 선택적으로, 처리 리소스의 선점과 해제를 수행하는 것은 코어 선점과 해제 모듈에서 실행하는 것이다. 상기 코

어 선점과 해제 모듈은 제1 운영체제 및/또는 제2 운영체제에서 실행되는 소프트웨어 모듈일 수 있다. 제1 운영체제 및 제2 운영체제에서 실행되는 것을 예로 들면, 코어 선점과 해제 모듈은 시스템 실시간 실행을 주로 완성하는 과정에서 이기종 운영체제에서 동적으로 프로세서 코어에 대한 선점과 해제를 수행하여, 프로세서 리소스 이용율의 최대화 목표에 달성한다. 하나의 운영체제의 리소스 선점의 성공은 다른 운영체제의 리소스 해제의 성공을 의미하며, 대응되게, 하나의 운영체제의 리소스 선점의 실패는 다른 운영체제의 리소스 해제의 실패를 의미한다. 도 5에 도시된 바와 같이, 코어 선점과 해제를 거쳐, (M+N)개 코어에서의 P개 코어를 실시간 운영체제에 스케줄링하고, (M+N-P)개 코어를 비실시간 운영체제에 스케줄링한다.

- [0169] 여기서, 코어 선점과 해제 모듈은 이기종 운영체제의 실시간 실행을 완성하는 과정에서, 이기종 운영체제에서 동적으로 프로세서 코어 등 처리 리소스에 대해 선점과 해제를 수행하여, 프로세서 리소스 이용율의 최대화 목표에 달성하도록 하기 위한 것이다.
- [0170] 본 실시예에서, 서로 다른 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 단계를 통해, 운영체제가 실행되는 과정에서 운영체제의 로드 상황에 기반하여 처리 리소스의 동적 조정을 수행할 수 있으며, 리소스 이용의 합리성을 향상시키고, 동시에 서비스 처리의 효율을 향상시킬 수 있다.
- [0171] 예시적인 일 실시예에서, 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 단계는 하기와 같은 단계 S101을 포함한다.
- [0172] S101에서, 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0173] 관련 기술에서 전용 가속 하드웨어에서 응답 속도에 대해 요구사항이 있는 서비스를 실행하는 방안의 경우, 프로세서와 가속 하드웨어(예를 들면, CPU와 FPGA/CPLD) 사이의 통신 버스 속도가 낮고, 전반 성능이 높지 않다. 본 실시예에서는, 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다. 즉, 하나의 운영체제에서 코어 간 통신 인터페이스를 통해 다른 운영체제로 처리 리소스의 선점을 위한 요청 또는 처리 리소스의 해제를 위한 요청을 발신하여, 처리 리소스의 선점 또는 해제를 실현할 수 있다. 통신 버스를 통해 인터랙션을 수행하는 방식에 비해, 프로세서 내부의 코어 간 통신 인터페이스를 통해 인터랙션을 수행하는 것은 더 높은 통신 속도를 가지며, 통신의 전반 성능을 향상시킨다.
- [0174] 예를 들면, 코어 간 통신 인터페이스 모듈은 제1 운영체제 및 제2 운영체제의 통신과 인터랙션 기능을 완성한다. 코어 선점과 해제를 수행하는 과정에서, 선점측과 해제측은 각자의 상태(예를 들면, 유휴 상태)를 코어 간 통신 인터페이스를 통해 상대측에 알릴 수 있으며, 동시에 특정의 신호 이벤트로 이기종 운영체제의 동작(예를 들면, 선점 동작)을 표시해야 한다. 따라서, 상기 코어 간 통신 인터페이스 모듈을 통해 코어 선점과 해제를 수행하여, 이기종 운영체제의 하드 코어 리소스 밸런스를 유지할 수 있고, 프로세서가 높은 점유율을 갖도록 보장할 수 있다.
- [0175] 선택적으로, 서로 다른 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 단계는, 예를 들면, SGI(Software Generated Interrupt, 소프트웨어에 의해 트리거되는 중단, Linux 시스템에서의 코어 간 중단)와 같은 코어 간 중단을 통해 완성될 수 있다. 하나의 운영체제는 IPI(Inter-Processor Interrupt, 프로세서 간 중단)를 통해 다른 운영체제로 리소스 선점 요청(예를 들면, 코어 선점 요청) 또는 리소스 해제 요청(예를 들면, 코어 해제 요청)을 발신하여, 처리 리소스의 선점 또는 해제를 수행할 수 있다. Linux 시스템을 예로 들면, Linux 시스템에서 사용자 정의된 중단 벡터 테이블 및 이기종 운영체제 간의 중단 이벤트에 기반하여 코어 간 통신을 실현할 수 있다. 여기서, IPI는 일반적인 주변기기 중단과는 다르게, SOC(System on Chip, 시스템 레벨 칩, 온칩 시스템이라고도 칭함) 내부의 복수의 코어(core) 사이에서 중단을 트리거하므로, 코어는 특별히 IPI에 일부분의 중단 번호를 예비하며, ARM64 아키텍처(CPU 아키텍처)에서는 0 내지 15의 16개 중단 번호이다.
- [0176] 선택적으로, 선점과 해제 과정은 이기종 운영체제의 실시간 로드 상황에 따라 결정된다. 예를 들면, 제2 운영체제의 로드가 급속히 증가되어, 더 많은 프로세서 코어의 지원이 필요할 경우, 이는 IPI를 통해 제1 운영체제로 코어 선점 요청을 발신하여, 제1 운영체제의 프로세서 코어를 선점할 수 있고, 이때 제1 운영체제가 유휴 상태(예를 들면, 태스크 스케줄링이 없음) 또는 적어도 일부분의 코어 리소스가 유휴 상태일 경우, 이는 제2 운영체제로 코어 리소스를 해제하며, 이때 제2 운영체제 코어 선점에 성공하고, 동시에 제1 운영체제의 코어 해제의 성공을 의미한다.
- [0177] 설명해야 할 것은, 코어 선점과 해제 과정은 하나의 동적으로 순환 반복되는 과정이다. 매회의 코어 선점과 해제는 모두 프로세서 하드 코어 리소스의 조정이 수반한다. 도 5에 도시된 바와 같이, 1회의 선점과 해제를 거쳐

코어세트1의 코어 수는 P로 되고, 코어세트2의 코어 수는 M+N-P로 된다. 또한, 코어 선점과 해제 과정은 로드 밸런스 전략, 서비스 관리, 리소스 적응형 스케줄링(예를 들면, 코어 적응형 스케줄링) 등 모듈과 협동하여 최종의 코어 조정 결과를 함께 결정할 수 있다.

- [0178] 여기서, 서로 다른 모듈에서 코어 조정 결과를 함께 결정하는 것은, 어느 한 시스템에서 더 많은 코어 리소스를 선점해야 할 경우, 최적적인 선점 성공 여부는 로드 밸런스, 서비스 관리, 코어 적응형 조정 등 모듈에서 함께 결정해야 하는 것을 가리킨다. 예를 들면, 그 중 한가지 가능한 상황에서, 제2 운영체제에서 더 많은 리소스가 필요할 경우, 이때 제1 운영체제가 유휴 상태(휴면 상태)이지만, 제1 운영체제에서 곧 하나의 중요한 서비스 스레드를 웨이크업하여 실행해야 하고, 또한 상기 스레드 실행 시 많은 리소스가 필요하므로, 이때 제1 운영체제에서는 제2 운영체제에 코어 리소스를 해제할 수 없다. 이 과정에서, 서비스 중요성을 판단하는 기준 및 최종적으로 리소스를 제2 운영체제에 할당하지 않는 동작 등은 모두 로드 밸런스 전략, 서비스 관리, 코어 적응형 조정 등 모듈의 참여에 관련된다. 이러한 모듈 각자의 전략은 개방성 특성을 포함하고, 이는 이들에 의해 함께 결정된 최종 핵심 조정 결과도 이러한 전략이 하계 작용한 결과임을 결정한다.
- [0179] 본 실시예에서, 코어 간 통신 인터페이스를 통해 서로 다른 운영체제 사이에서의 처리 리소스의 선점과 해제를 수행하는 단계를 통해, 통신 속도를 향상시킬 수 있고, 통신의 전반 성능을 향상시킬 수 있다.
- [0180] 예시적인 일 실시예에서는, 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 단계는 하기와 같은 단계 S111 및 S112를 포함한다.
- [0181] S111에서, 코어 간 통신 인터페이스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송한다. 여기서, 제1 인터랙션 요청은 제2 운영체제와 리소스 인터랙션을 수행하는 것을 요청하기 위한 것이고, 리소스 인터랙션은 리소스 선점 및 리소스 해제 중의 하나를 포함한다.
- [0182] S112에서, 코어 간 통신 인터페이스를 통해 제2 운영체제에서 제1 인터랙션 요청에 응답하여 반환하는 제1 인터랙션 응답을 획득한다. 여기서, 제1 인터랙션 응답은 제1 운영체제에서 제1 인터랙션 응답에 따라 제2 운영체제와 리소스 인터랙션을 수행하는 것을 지시하기 위한 것이다.
- [0183] 본 실시예에서는, 제1 운영체제의 경우, 프로세서에 의해 처리되는 대부분 서비스는 서비스 응답 속도에 대한 요구사항이 높지 않은 서비스인 것을 고려하여, 대응되게 제1 운영체제에 의해 처리되는 서비스의 서비스량이 일반적으로 적으므로, 제1 운영체제는 코어 간 통신 인터페이스를 통해 제2 운영체제의 처리 리소스를 선점하거나 능동적으로 제2 운영체제로 그에 의해 점유된 처리 리소스를 해제할 것을 요청할 수 있다. 선택적으로, 처리 리소스의 선점과 해제는 요청-응답의 인터랙션 방식을 통해 수행될 수 있다.
- [0184] 제2 운영체제의 처리 리소스를 선점해야 하거나 능동적으로 제2 운영체제로 처리 리소스를 해제해야 할 경우, 제1 운영체제에서는 코어 간 통신 인터페이스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송하여, 제2 운영체제와 리소스 선점과 리소스 해제 중의 한가지 리소스 인터랙션을 수행하는 것을 요청할 수 있다. 제1 인터랙션 요청을 수신한 후, 제2 운영체제에서는 자체의 로드 상황 등에 기반하여 제1 운영체제에서 적어도 일부분의 처리 리소스를 점유하는 것을 허용할지 여부 또는 제1 운영체제에 의해 해제된 처리 리소스를 수신할지 여부를 결정할 수 있고, 코어 간 통신 인터페이스를 통해 제1 인터랙션 응답을 제1 운영체제로 반환하여, 제1 운영체제에서 제1 인터랙션 응답에 따라 제2 운영체제와 리소스 인터랙션을 수행하도록 할 수 있다.
- [0185] 설명해야 할 것은, 상기 처리 리소스의 선점과 해제는 운영체제 분야에서의 처리 리소스의 스케줄링(즉, 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스 스케줄링의 협상을 완성함)일 뿐, 실제적인 처리 리소스의 스케줄링은 아니며, 처리 리소스의 실제적인 스케줄링을 수행할 시, 리소스 적응형 스케줄링 모듈에서 처리 리소스의 실제적인 스케줄링을 완성한다.
- [0186] 본 실시예에서, 요청-응답의 방식을 통해 서로 다른 운영체제 사이의 리소스 인터랙션을 수행하고, 또한 응답 속도가 빠른 운영체제에서 응답 속도가 느린 운영체제로 능동적으로 처리 리소스를 해제하는 것을 허용하는 단계를 통해, 처리 리소스의 이용율을 향상시킬 수 있다.
- [0187] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S121 및 S122를 더 포함한다.
- [0188] S121에서, 제1 운영체제의 처리 리소스의 리소스 이용율을 검출한다.
- [0189] S122에서, 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 제1 운영체제에서 제2 운영체제와 리소스 인터랙션을 수행하는 것을 대기하는 것으로 결정할 경우, 코어 간 통신 인터랙션을 통해 제1 인터랙션 요청을 제2

운영체제에 전송하는 것을 트리거한다.

- [0190] 본 실시예에서는, 처리 리소스의 선점과 해제 과정은 이기종 운영체제의 실시간 로드 상황에 따라 결정될 수 있다. 서로 다른 운영체제 내에서는 시스템의 로드 상황을 검출하기 위한 로드 검출 모듈이 실행될 수 있으며, 이는 본 시스템 내의 시스템 제어 모듈에 속할 수 있다. 제1 운영체제의 경우, 상기 제1 운영체제에는 제1 시스템 제어 모듈이 실행될 수 있으며, 상기 시스템 제어 모듈에서의 로드 검출 모듈은 제1 운영체제의 처리 리소스의 리소스 이용율을 검출할 수 있다. 제1 운영체제의 처리 리소스의 리소스 이용율은 제1 운영체제의 로드 상황을 나타내는데 사용될 수 있으며, 처리 리소스의 리소스 이용율이 높을 수록, 시스템 로드는 더 가중하고, 반대로 처리 리소스의 리소스 이용율이 낮을 수록, 시스템 로드는 더 가볍다.
- [0191] 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 제1 운영체제에서 제2 운영체제와 리소스 인터랙션을 수행해야 할지 여부를 결정할 수 있다. 예를 들면, 하나의 운영체제(예를 들면, 제1 운영체제)의 현재의 리소스 이용율이 너무 높거나(지정된 이용율 임계값에 도달함), 또는 하나의 운영체제에서 실행 대상인 서비스에 따라 상기 운영체제의 미래의 리소스 이용율이 너무 높은(지정된 이용율 임계값에 도달함) 것으로 결정할 경우, 상기 운영체제에서 다른 운영체제(예를 들면, 제2 운영체제)의 처리 리소스를 선점해야 하는 것으로 결정할 수 있고, 리소스 이용율이 너무 낮을 경우(다른 지정된 이용율 임계값보다 낮음), 상기 운영체제에서 다른 운영체제로 처리 리소스를 해제해야 하는 것으로 결정할 수 있다. 선택적으로, 제1 운영체제에서 제2 운영체제와 리소스 인터랙션을 수행하는 것을 대기하는 것으로 결정할 경우, 코어 간 통신 인터페이스를 통해 제1 인터랙션 요청을 제2 운영체제에 전송하는 단계를 실행하는 것을 트리거할 수 있다.
- [0192] 본 실시예에서, 요청-응답의 방식을 통해 서로 다른 운영체제 사이의 리소스 인터랙션을 수행하고, 또한 응답 속도가 빠른 운영체제에서 응답 속도가 느린 운영체제로 능동적으로 처리 리소스를 해제하는 것을 허용하는 단계를 통해, 처리 리소스의 이용율을 향상시킬 수 있다.
- [0193] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S131 및 S132 중의 적어도 한 단계를 더 포함한다.
- [0194] S131에서, 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 제1 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0195] S132에서, 제1 운영체제의 처리 리소스가 유휴 상태이고, 제1 운영체제에서 실행 대상인 서비스가 존재하지 않을 경우, 제1 운영체제에서 제2 운영체제로 처리 리소스를 해제하는 것을 대기하는 것으로 결정한다.
- [0196] 본 실시예에서는, 제1 운영체제의 처리 리소스의 이용율의 임계값, 즉 제1 이용율 임계값은 사전 설정될 수 있다. 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 제1 운영체제의 현재의 로드가 너무 가중하여, 제2 운영체제의 처리 리소스를 선점하여 자체의 서비스 수요를 충족시켜야 하는 것으로 결정할 수 있다.
- [0197] 선택적으로, 제1 운영체제의 응답 속도는 제2 운영체제보다 빠르고, 일반적으로 특정 서비스를 위한 것이며, 특정 서비스의 트리거는 소정의 랜덤성을 갖고, 통상적으로 서비스 빈도가 낮으므로, 제1 운영체제의 처리 리소스는 소정 확율이 유휴 상태이고, 또한 제1 운영체제에 의해 실행 대상인 서비스가 존재하지 않는다. 이러한 경우에는, 제1 운영체제에서 제2 운영체제에 적어도 일부분의 처리 리소스를 해제하는 것을 대기하는 것으로 결정할 수 있다.
- [0198] 여기서, 제1 운영체제의 처리 리소스가 유휴 상태일 경우, 제1 운영체제와 대응되는 실행 대상인 서비스 큐를 검색할 수 있다. 상기 실행 대상인 서비스 큐가 비어있을 경우(즉, 실행 대상인 서비스가 없음), 제1 운영체제가 소정 시간 범위 내에서 실행 대상인 서비스가 존재하지 않는 것으로 결정하고, 실행 대상인 서비스 큐가 비어있지 않을 경우(즉, 실행 대상인 서비스를 포함함), 제1 운영체제에서 현재 서비스를 실행하지 않아도, 제2 운영체제에 처리 리소스를 해제하지 않는다.
- [0199] 본 실시예에서, 처리 리소스의 리소스 이용율이 너무 높을 경우 처리 리소스를 선점하며, 처리 리소스가 유휴 상태이고 후속적으로 처리해야 할 서비스가 없을 경우 처리 리소스를 해제하는 단계를 통해, 리소스 스케줄링의 합리성을 향상시킬 수 있고, 처리 리소스의 이용율을 향상시킬 수 있다.
- [0200] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S141을 더 포함한다.
- [0201] S141에서, 제1 운영체제에는 스케줄링할 서비스가 없고, 제1 운영체제에 할당 대상인 서비스가 없을 경우, 제1 운영체제가 휴면 상태에 진입하도록 제어한다.

- [0202] 본 실시예에서는, 제1 운영체제가 프로세서에 기반하여 주기적으로 실행될 수 있거나, 또는 제1 운영체제가 수신된 웨이크업 요청에 응답하여 프로세서에 기반하여 실행될 수 있거나, 또는 제1 운영체제가 프로세서에서 생성되는 현재 운영체제와 제1 운영체제 사이의 매칭도에 따라 프로세서에 기반하여 실행될 수 있다. 제1 운영체제는 실행 종료 후 휴면될 수 있고, 제2 운영체제는 제1 운영체제가 휴면하는 동안 제1 운영체제에서 사용되는 프로세서 리소스(예를 들면, 프로세서 코어)를 제2 운영체제의 스케줄링 리소스풀(즉, 사용 가능한 리소스풀)에 추가할 수 있다.
- [0203] 선택적으로, 제1 운영체제에는 스케줄링할 서비스가 없고, 제1 운영체제에 할당 대상인 서비스(예를 들면, 제1 운영체제의 리소스 점유율이 이에 설정된 점유율의 하한보다 낮거나 또는 곧 낮아짐)이 없을 경우, 제1 운영체제의 능동적인 휴면을 트리거할 수 있다. 즉, 능동적으로 휴면 상태에 진입할 수 있다. 제1 운영체제가 휴면 상태에 진입하도록 제어하는 단계 이전에, 먼저 제2 운영체제에 제1 운영체제의 코어 리소스를 해제할 수 있고(중단 요청의 방식을 통해 리소스를 해제할 수 있음), 제1 운영체제의 시스템 현장 데이터를 저장할 수도 있다. 예를 들면, 제1 운영체제가 웨이크업된 후, 저장된 시스템 현장 데이터에 기반하여 이전의 실행 상태로 복구되도록, 상기 데이터를 스택에 압입한다.
- [0204] 예를 들면, 제1 운영체제에 스케줄링할 태스크가 없을 경우(즉, 유향 상태), 제1 운영체제에서 코어 해제 요청을 제2 운영체제에 발신한다. 이때, 제1 운영체제의 코어 리소스는 제2 운영체제에서 연결 관리하며, 제1 운영체제는 휴면 상태에 진입한다.
- [0205] 선택적인 일 응용 장면에서는, 제1 운영체제(예를 들면, RTOS)는 주기적으로 웨이크업되어 실행되고, 제1 운영체제와 제2 운영체제는 동일한 처리 리소스(예를 들면, CPU 코어가 0인 프로세서 코어일 수 있음)를 교체 점유하여 스케줄링하며, 상기 동일한 프로세서 리소스는 제1 처리 리소스일 수 있고, 대응되게 제2 운영체제의 처리 리소스는 제2 처리 리소스일 수 있다. 제1 운영체제에서 제1 처리 리소스를 스케줄링하는 타임 슬라이스 내에서, 제2 운영체제는 제1 시각에 상기 제1 처리 리소스를 연결 관리하도록 하나의 중단을 생성하여, 제1 운영체제가 휴면해야만 한다. 이때, 제1 운영체제는 현장을 스택에 저장하고 휴면하며, 다음 제1 처리 리소스를 제2 운영체제에 해제하여 연결 관리하도록 하고, 제2 운영체제에서 스케줄링 완성 후, 제2 시각에 제1 운영체제에서 제1 처리 리소스를 선점하도록 중단을 생성하여 제1 운영체제를 웨이크업하며, 제1 운영체제는 상기 시각으로부터 다시 순환 모드에 진입하여 제1 처리 리소스를 점유하여 스케줄링을 시작한다.
- [0206] 상기 선택적인 응용 장면에서는, 제2 운영체제에서 제1 처리 리소스를 스케줄링하는 타임 슬라이스 내에서, 제1 운영체제는 휴면 상태이고, 제3 시각에 제1 운영체제는 하드웨어에 의해 보고된 중단 이벤트로 인해 웨이크업될 수 있으며, 제2 운영체제는 제1 처리 리소스에서 실행되는 프로세스 현장을 예비하고, 제1 운영체제에서 제1 처리 리소스를 점유하며, 하드웨어에 의해 보고된 중단 이벤트를 처리 완료 후, 제4 시각에 다시 휴면 상태에 진입한다. 이때, 제1 운영체제에서는 제2 운영체제에 제1 처리 리소스의 중단을 해제하는 것을 보고하고, 제2 운영체제에서는 계속 설정된 주기에 따라 제1 처리 리소스를 스케줄링하여, 현장 실행 프로세스를 복구한다.
- [0207] 선택적으로, 본 실시예에서는, 제1 운영체제가 주기적으로 실행될 경우, 단일 주기의 시간과 2개의 실행 주기 사이 간격의 시간은 동일할 수도 있고 서로 다를 수도 있다. 2개의 실행 주기 사이 간격의 시간 내에 제1 운영체제는 휴면 상태일 수 있지만, 이에 한정되지 않는다. 제1 운영체제에 할당된 처리 리소스는 제2 운영체제에 의해 사용된다. 단일 실행 주기의 시간이 2개의 실행 주기 사이 간격의 시간과 동일할 경우, 제1 운영체제에 할당된 처리 리소스의 동일한 시간을 제1 운영체제와 제2 운영체제에서 교체 점유한다. 단일 실행 주기의 시간이 2개의 실행 주기 사이 간격의 시간과 서로 다를 경우, 제1 운영체제에 할당된 처리 리소스의 서로 다른 시간을 제1 운영체제와 제2 운영체제에서 교체 점유한다. 제1 운영체제에서 점유한 시간은 제2 운영체제에서 점유한 시간보다 클 수 있고, 제2 운영체제에서 점유한 시간은 제1 운영체제에서 점유한 시간보다 클 수도 있다.
- [0208] 선택적인 일 실시형태에서는, 기기가 제1 운영체제 웨이크업을 트리거하여 실행되는 것을 예로 들어, 트리거 모드에서 제1 운영체제(예를 들면, RTOS) 웨이크업 전략을 제공한다. 트리거 모드는 제1 운영체제 버스 도메인의 기기에서 발신되는 중단에 의해 시작될 수 있다. 제1 운영체제 버스 도메인에는 기기0 내지 기기N이 연결되며, 제1 운영체제가 휴면 상태일 경우, 어느 한 시각에 기기0이 제1 운영체제에 중단을 트리거한다고 가정하면, 제1 운영체제는 즉시로 웨이크업되고, 웨이크업된 제1 운영체제는 제2 운영체제에 제1 처리 리소스의 중단을 선점하는 것을 먼저 트리거하며, 제2 운영체제는 중단을 수신한 후 제1 처리 리소스를 먼저 해제하여, 현장을 저장(실행되는 데이터를 스택에 압입함)한 다음, 제1 운영체제에서 제1 처리 리소스를 스케줄링하여 기기0에 의해 트리거된 중단이 지시하는 동작 서비스를 처리한다. 현재 폴링 모드일 경우, 후속 처리 과정은 상기 폴링 모드와 동일하며, 여기에서 설명을 생략한다.

- [0209] 본 실시예에서, 하나의 운영체제에서 다른 운영체제에 전부의 처리 리소스(예를 들면, 코어 리소스)를 해제하고, 점유된 처리 리소스가 0일 경우, 운영체제가 휴면 상태에 진입하도록 제어하는 단계를 통해, 운영체제에서 다른 운영체제에 전부의 처리 리소스를 해제하는 것을 허용함으로써, 처리 리소스의 이용율을 향상시킨다.
- [0210] 예시적인 일 실시예에서는, 코어 간 통신 인터페이스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송하는 단계는 하기와 같은 단계 S151 및 S152를 포함한다.
- [0211] S151에서, 코어 간 통신 인터페이스를 통해 중단 번호가 제1 중단 번호인 제1 선점 요청을 제2 운영체제에 전송한다. 여기서, 제1 선점 요청은 제2 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0212] S152에서, 코어 간 통신 인터페이스를 통해 중단 번호가 제2 중단 번호인 리소스 해제 요청을 제2 운영체제에 전송한다. 여기서, 리소스 해제 요청은 제2 운영체제에 제1 운영체제에 의해 점유된 처리 리소스를 해제하는 것을 요청하기 위한 것이다.
- [0213] 본 실시예에서는, 서로 다른 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것은 코어 간의 중단을 통해 완성될 수 있고, 서로 다른 중단 이벤트가 서로 다른 리소스에 대한 인터랙션 유형을 정의할 수 있다. 서로 다른 중단 이벤트는 서로 다른 중단 번호에 대응될 수 있다. 여기서, 제1 운영체제에서 제2 운영체제로 처리 리소스를 선점하는 것을 요청하는 제1 중단 이벤트에 할당되는 중단 번호는 제1 중단 번호이고, 제1 운영체제에서 능동적으로 제2 운영체제로 처리 리소스를 해제하는 제2 중단 이벤트에 할당되는 중단 번호는 제2 중단 번호이다.
- [0214] 선택적인 일 실시형태로서, 제1 인터랙션 요청은 제1 선점 요청일 수 있다. 즉 제2 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다. 제1 운영체제는 코어 간 통신 인터페이스를 통해 중단 번호가 제1 중단 번호인 제1 선점 요청을 제2 운영체제에 전송할 수 있다. 즉, 중단 번호가 제1 중단 번호인 코어 간 통신 중단의 트리거 소스는 제1 운영체제이고, 응답 소스는 제2 운영체제이며, 상기 코어 간 통신 중단은 제1 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 의미한다.
- [0215] 선택적인 다른 일 실시형태에서는, 제1 인터랙션 요청은 리소스 해제 요청일 수 있다. 즉, 제1 운영체제에 의해 점유된 처리 리소스를 제2 운영체제에 해제하는 것을 요청하기 위한 것이고, 제1 운영체제는 코어 간 통신 인터페이스를 통해 중단 번호가 제2 중단 번호인 리소스 해제 요청을 제2 운영체제에 전송할 수 있다. 즉 중단 번호가 제2 중단 번호인 코어 간 통신 중단의 트리거 소스는 제1 운영체제이고, 응답 소스는 제2 운영체제이며, 상기 코어 간 통신 중단은 제1 운영체제에서 제2 운영체제에 처리 리소스를 능동적으로 해제하는 것을 의미한다.
- [0216] 본 실시예에서, 서로 다른 운영체제 사이에서 처리 리소스에 대한 선점과 능동적 해제를 위한 중단 번호를 할당하고, 코어 간 통신 중단의 방식을 통해 운영체제 간의 리소스 스케줄링을 수행하는 단계를 통해, 리소스 스케줄링의 정확성을 향상시킨다.
- [0217] 예시적인 일 실시예에서는, 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 단계는 하기와 같은 단계 S161 및 S162를 포함한다.
- [0218] S161에서, 코어 간 통신 인터페이스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송한다. 여기서, 제2 인터랙션 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0219] S162에서, 코어 간 통신 인터페이스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득한다. 여기서, 제2 인터랙션 응답은 제1 운영체제가 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는지 여부를 지시하기 위한 것이다.
- [0220] 본 실시예에서는, 제2 운영체제의 경우, 프로세서에 의해 처리되는 대부분의 서비스가 서비스 응답 속도에 대한 요구사항이 높지 않은 서비스고, 대응되게, 제2 운영체제에 의해 처리되는 서비스의 서비스량이 통상적으로 많은 것을 고려하여, 제2 운영체제는 코어 간 통신 인터페이스를 통해 제1 운영체제의 처리 리소스를 선점하는 것을 요청할 수 있다. 선택적으로, 처리 리소스의 선점은 요청-응답의 인터랙션 방식을 통해 수행될 수 있다.
- [0221] 제1 운영체제의 처리 리소스를 선점해야 할 경우, 제2 운영체제에서는 코어 간 통신 인터페이스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송하여, 제1 운영체제와 리소스 선점을 목적으로 하는 리소스 인터랙션을 수행하도록 요청할 수 있다. 제2 인터랙션 요청을 수신한 후, 제1 운영체제에서는 자체의 로드 상황 등에 기반하여 제2 운영체제에서 그의 적어도 일부분의 처리 리소스를 점유하는 것을 허용하는지 여부를 결정할 수 있으며, 코어 간 통신 인터페이스를 통해 제2 인터랙션 응답을 제2 운영체제에 반환할 수 있어, 제2 운영체

제에서 제2 인터랙션 응답에 따라 제1 운영체제와 리소스 인터랙션을 수행하도록 할 수 있다.

- [0222] 설명해야 할 것은, 상기 처리 리소스의 선점은 운영체제 차원에서의 처리 리소스에 대한 스케줄링(즉, 제1 운영체제와 제2 운영체제 사이에서 처리 리소스 스케줄링의 협상을 완성함)이며, 처리 리소스에 대한 실제적인 스케줄링은 아니다. 처리 리소스에 대해 실제적인 스케줄링을 수행할 경우, 리소스 적응형 스케줄링 모듈에서 처리 리소스에 대한 실제적인 스케줄링을 완성한다.
- [0223] 본 실시예에서, 요청-응답의 방식을 통해 서로 다른 운영체제 사이의 리소스 인터랙션을 수행하는 단계를 통해, 처리 리소스의 이용율을 향상시킬 수 있고, 동시에 서비스 처리의 응답 속도를 향상시킬 수 있다.
- [0224] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S171 및 S172를 더 포함한다.
- [0225] S171에서, 제2 운영체제의 처리 리소스의 리소스 이용율을 검출한다.
- [0226] S172에서, 제2 운영체제의 처리 리소스의 리소스 이용율에 따라, 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정할 경우, 코어 간 통신 인터페이스를 통해 제2 인터랙션 요청을 제1 운영체제에 전송한다.
- [0227] 본 실시예에서는, 처리 리소스의 선점 과정은 이기종 운영체제의 실시간 로드 상황에 의해 결정될 수 있다. 서로 다른 운영체제 내에서는 시스템의 로드 상황을 검출하기 위한 로드 검출 모듈이 실행될 수 있으며, 이는 본 시스템 내의 시스템 제어 모듈에 속할 수 있다. 제2 운영체제의 경우, 상기 제2 운영체제에는 제2 시스템 제어 모듈이 실행될 수 있으며, 상기 시스템 제어 모듈에서의 로드 검출 모듈은 제2 운영체제의 처리 리소스의 리소스 이용율을 검출할 수 있다. 제2 운영체제의 처리 리소스의 리소스 이용율은 제2 운영체제의 로드 상황을 나타내는데 사용될 수 있으며, 처리 리소스의 리소스 이용율이 높을 수록, 시스템 로드는 더 가중하고, 반대로 처리 리소스의 리소스 이용율이 낮을 수록, 시스템 로드는 더 가볍다.
- [0228] 제2 운영체제의 처리 리소스의 리소스 이용율에 따라, 제2 운영체제에서 제1 운영체제와 리소스 인터랙션을 수행해야 할지 여부를 결정할 수 있다. 예를 들면, 현재의 리소스 이용율이 너무 높거나(지정된 이용율 임계값에 도달함), 또는 실행 대상인 서비스에 따라 미래의 리소스 이용율이 너무 높은(지정된 이용율 임계값에 도달함) 것으로 결정할 경우, 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점해야 하는 것으로 결정할 수 있다. 제2 운영체제에서 제1 운영체제와 리소스 인터랙션을 수행하는 것을 대기하는 것으로 결정할 경우, 코어 간 통신 인터페이스를 통해 제2 인터랙션 요청을 제1 운영체제에 전송하는 단계를 실행하는 것을 트리거할 수 있다.
- [0229] 본 실시예에서, 요청-응답의 방식을 통해 서로 다른 운영체제 사이의 리소스 인터랙션을 수행하고, 또한 응답 속도가 느린 운영체제에서 응답 속도가 빠른 운영체제로 능동적으로 처리 리소스를 선점하는 것을 허용하는 단계를 통해, 처리 리소스의 이용율을 향상시킬 수 있다.
- [0230] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S181 및 S182 중의 적어도 한 단계를 더 포함한다.
- [0231] S181에서, 제2 운영체제의 현재의 리소스 이용율이 제2 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0232] S182에서, 제2 운영체제의 실행 중 서비스 및 실행 대상인 서비스에 따라 제2 운영체제의 처리 리소스의 리소스 이용율이 제3 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0233] 본 실시예에서는, 서비스 처리의 복잡성을 고려하여, 서비스와 서비스 사이에는 결합 관계가 존재할 수 있으며, 서비스의 중요성은 그가 할당을 허용하는 운영체제에 영향준다. 또한, 서로 다른 서비스 사이의 우선 처리 등급 사이에도 구별점이 존재할 수 있다. 제2 운영체제는 자체의 리소스 이용율, 그의 할당 대상인 서비스, 실행 대상인 서비스 등 복수 개 측면의 요소를 종합하여, 제1 운영체제의 처리 리소스를 선점해야 할지 여부를 결정할 수 있다.
- [0234] 선택적인 일 실시형태로서, 제2 운영체제의 처리 리소스의 리소스 이용율의 임계값, 즉 제2 이용율 임계값은 사전 설정될 수 있다. 제2 운영체제의 처리 리소스의 리소스 이용율이 제2 이용율 임계값 이상인 경우, 제2 운영체제의 현재의 로드가 너무 가중하여, 제1 운영체제의 처리 리소스를 선점하여 자체의 서비스 수요를 충족시켜야 하는 것으로 결정할 수 있다.
- [0235] 선택적인 다른 일 실시형태에서, 현재 실행 중인 서비스 외에, 제2 운영체제는 제2 운영체제에 할당된 서비스도 처리해야 하므로, 제2 운영체제의 처리 리소스의 리소스 이용율의 임계값, 즉 제3 이용율 임계값은 사전 설정될

수 있다. 제2 운영체제의 실행 중 서비스 및 제2 운영체제의 실행 대상인 서비스에 따라 제2 운영체제의 처리 리소스의 리소스 이용율이 제3 이용율 임계값 이상인 것으로 결정할 경우, 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정할 수 있고, 제2 운영체제는 곧 로드가 가중한 상태로 되어, 제1 운영체제의 처리 리소스를 선점하여 자체의 서비스 수요를 충족시켜야 하는 것으로 결정할 수 있다.

[0236] 또한, 할당된 서비스(현재 실행 중인 서비스를 포함할 수 있고, 할당된 서비스를 포함할 수도 있음)을 실행하는 외에, 제2 운영체제는 현재 할당 대상인 서비스(즉, 새로 추가된 할당 대상인 서비스)에서 그의 할당된 서비스와의 트랜잭션 결합도가 높은 서비스를 처리 할 수도 있으므로, 제2 운영체제의 처리 리소스의 리소스 이용율의 임계값, 즉 제4 이용율 임계값 및 새로 추가된 할당 대상인 서비스와 할당된 서비스 사이 서비스 결합도의 임계값, 즉 제3 결합도 임계값은 사전 설정될 수 있다. 제2 운영체제의 처리 리소스의 현재의 리소스 이용율이 제4 이용율 임계값 이상이고, 또한 새로 추가된 할당 대상인 서비스가 제2 운영체제의 할당된 서비스와의 서비스 결합도가 제3 결합도 임계값 이상인 경우, 제2 운영체제는 곧 로드가 가중한 상태로 되어, 제1 운영체제의 처리 리소스를 선점하여 자체의 서비스 수요를 충족시켜야 하는 것으로 결정할 수 있다.

[0237] 선택적으로, 제2 이용율 임계값과 제3 이용율 임계값은, 예를 들면, 양자가 모두 50%, 60% 또는 다른 값인 것과 같이 동일할 수도 있고, 서로 다를 수도 있으며, 제4 이용율 임계값은, 예를 들면, 40%와 같이 제2 이용율 임계값 및 제3 이용율 임계값보다 작은 하나의 값일 수도 있고, 다른 이용율 임계값일 수도 있다. 본 실시예에서는 각 이용율 임계값에 대해 한정하지 않는다.

[0238] 본 실시예에서, 운영체제의 현재 실행 중 서비스, 현재 실행 중 서비스와 할당된 서비스, 또는 현재 실행 중 서비스와 할당된 서비스 및 새로 추가된 할당 대상인 서비스에 기반하여, 운영체제에 대한 리소스 스케줄링 수요를 결정하는 단계를 통해, 리소스 스케줄링의 원활성을 향상시킬 수 있고, 동시에 처리 리소스의 이용율을 향상시킬 수도 있다.

[0239] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S191 및 S192를 더 포함한다.

[0240] S191에서, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮지 않을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것으로 결정한다.

[0241] S192에서, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것으로 결정한다.

[0242] 제1 운영체제의 경우, 상기 운영체제가 현재 유휴 상태이고, 실행될 할당된 서비스가 없거나, 또는 제1 운영체제에 할당해야 할 할당 대상인 서비스가 없을 경우, 제1 운영체제에서는 제2 운영체제에서 자체의 처리 리소스를 선점하는 것을 허용하는 것으로 직접 결정할 수 있다. 제1 운영체제에서 현재 서비스를 처리 중에 있거나, 할당된 실행 대상인 서비스가 존재하거나, 또는 제1 운영체제에 할당해야 할 할당 대상인 서비스가 존재할 경우, 상기 서비스의 서비스 우선순위 등 요소를 종합하여 제2 운영체제에서 제1 운영체제의 처리 리소스를 점유하는 것을 허용할지 여부를 결정해야 한다.

[0243] 선택적으로, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮지 않을 경우, 제1 운영체제의 실행 중인 서비스의 긴급 정도가 제2 운영체제의 실행 대상인 서비스의 긴급 정도보다 낮지 않고, 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것으로 결정을 나타낸다. 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 실행 대상인 서비스의 서비스 우선순위보다 낮을 경우, 제1 운영체제의 실행 중인 서비스의 긴급 정도가 제2 운영체제의 실행 대상인 서비스의 긴급 정도보다 낮고, 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것으로 결정함을 나타낸다.

[0244] 여기서, 제1 운영체제의 응답 속도가 제2 운영체제의 응답 속도보다 빠르므로, 통상적으로 서비스 우선순위가 높은 서비스는 우선적으로 제1 운영체제에 할당되지만, 서비스 사이의 결합 관계를 고려하여, 서비스의 우선순위가 높은 서비스가 제2 운영체제에 할당될 수도 있다. 따라서, 리소스를 선점할 시 서비스의 우선순위에 기반하여 판단을 수행할 수 있다. 제1 운영체제의 실행 중인 서비스 및 실행 대상인 서비스의 서비스 우선순위가 모두 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮지 않을 경우에만, 처리 리소스에 대한 선점을 허용하는 것을 고려할 수도 있다.

[0245] 본 실시예에서, 서비스 우선순위 등 요소를 종합하여 운영체제 사이의 처리 리소스에 대한 스케줄링을 수행하는

단계를 통해, 리소스 스케줄링의 원활성 및 합리성을 향상시킬 수 있다.

- [0246] 예시적인 일 실시예에서는, 코어 간 통신 인터페이스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송하는 단계는 하기와 같은 단계 S201를 포함한다.
- [0247] S201에서, 코어 간 통신 인터페이스를 통해 중단 번호가 제3 중단 번호인 제2 선점 요청을 제1 운영체제에 전송한다. 여기서, 제2 선점 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0248] 본 실시예에서는, 상기 실시예와 유사하게, 서로 다른 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것은 코어 간의 중단을 통해 완성될 수 있고, 서로 다른 중단 이벤트가 서로 다른 리소스에 대한 인터랙션 유형을 정의할 수 있다. 서로 다른 중단 이벤트는 서로 다른 중단 번호에 대응될 수 있다. 여기서, 제2 운영체제에서 제1 운영체제로 처리 리소스를 선점하는 것을 요청하는 제3 중단 이벤트에 할당되는 중단 번호는 제3 중단 번호이다.
- [0249] 선택적으로, 제2 인터랙션 요청은 제2 선점 요청일 수 있다. 즉 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다. 제2 운영체제는 코어 간 통신 인터페이스를 통해 중단 번호가 제3 중단 번호인 제2 선점 요청을 제1 운영체제에 전송할 수 있다. 즉, 중단 번호가 제3 중단 번호인 코어 간 통신 중단의 트리거 소스는 제2 운영체제이고, 응답 소스는 제1 운영체제이며, 상기 코어 간 통신 중단은 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 의미한다.
- [0250] 본 실시예에서, 서로 다른 운영체제 사이에서 처리 리소스에 대한 선점을 위한 중단 버호를 할당하고, 코어 간 통신 중단의 방식을 통해 운영체제 간의 리소스 스케줄링을 수행하는 단계를 통해, 리소스 스케줄링의 정확성을 향상시킨다.
- [0251] 예시적인 일 실시예에서는, 코어 간 통신 인터페이스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하는 단계는 하기와 같은 단계 S211 및 S212 중의 적어도 한 단계를 포함한다.
- [0252] S211에서, 코어 간 통신 인터페이스를 통해 중단 번호가 제4 중단 번호인 리소스 해제를 허용하는 응답을 제2 운영체제에 전송한다. 여기서, 리소스 해제를 허용하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것을 지시하기 위한 것이다.
- [0253] S212에서, 코어 간 통신 인터페이스를 통해 중단 번호가 제5 중단 번호인 리소스 해제를 거부하는 응답을 제2 운영체제에 전송한다. 여기서, 리소스 해제를 거부하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것을 지시하기 위한 것이다.
- [0254] 본 실시예에서는, 상기 실시예와 유사하게, 서로 다른 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것은 코어 간의 중단을 통해 완성될 수 있고, 서로 다른 중단 이벤트가 서로 다른 리소스에 대한 인터랙션 유형을 정의할 수 있다. 서로 다른 중단 이벤트는 서로 다른 중단 번호에 대응될 수 있다. 여기서, 제2 운영체제에서 제2 운영체제로 리소스 선점 응답을 반환하는 것을 요청하는 제4 중단 이벤트 및 제5 중단 이벤트에 할당되는 중단 번호는 제5 중단 번호이며, 제4 중단 이벤트는 리소스 선점을 허용하는 중단 이벤트이고, 제5 중단 이벤트는 리소스 선점을 거부하는 중단 이벤트이다.
- [0255] 선택적인 일 실시형태로서, 제1 운영체제가 제2 운영체제에서 그의 적어도 일부분의 처리 리소스를 점유하는 것을 허용하는 것으로 결정할 경우, 제1 운영체제에서는 코어 간 통신 인터페이스를 통해 중단 번호가 제4 중단 번호인 리소스 해제를 허용하는 응답을 제2 운영체제에 전송할 수 있다. 즉, 중단 번호가 제4 중단 번호인 코어 간 통신 중단의 트리거 소스는 제1 운영체제이고, 응답 소스는 제2 운영체제이며, 상기 코어 간 통신 중단은 제1 운영체제에서 제2 운영체제에 처리 리소스를 수동적으로 해제하는 것을 의미한다.
- [0256] 선택적인 다른 일 실시형태로서, 제1 운영체제가 제2 운영체제에서 그의 처리 리소스를 점유하는 것을 거부하는 것으로 결정할 경우, 제1 운영체제에서는 코어 간 통신 인터페이스를 통해 중단 번호가 제5 중단 번호인 리소스 해제를 거부하는 응답을 제2 운영체제에 전송할 수 있다. 즉, 중단 번호가 제5 중단 번호인 코어 간 통신 중단의 트리거 소스는 제1 운영체제이고, 응답 소스는 제2 운영체제이며, 상기 코어 간 통신 중단은 제1 운영체제에서 제2 운영체제에 처리 리소스를 해제하는 것을 거부하는 것을 의미한다.
- [0257] 본 실시예에서, 서로 다른 운영체제 사이에서 처리 리소스 선점을 허용 및 처리 리소스 선점을 거부하기 위한 중단 번호를 할당하고, 코어 간 통신 중단의 방식을 통해 운영체제 간의 리소스 스케줄링을 수행하는 단계를 통

해, 리소스 스케줄링의 정확성을 향상시킨다.

[0258] 선택적인 일 실시예로서, 아래에서는 RTOS 시스템 및 Linux 시스템을 예로 들어 코어 간 통신 중단을 해석하여 설명한다. Linux 시스템에서는 사용자 정의된 중단 벡터 테이블 및 이기종 운영체제 간의 중단 이벤트에 기반하여 코어 간 통신을 실현할 수 있다. Linux 운영체제는 SGI 정의되지 않은 비트 번호를 충분히 이용하여, 단말기 신호의 사용자 정의를 실현하고, 코어 간 통신의 비용을 감소한다. 정의되지 않은 비트 번호는 8번 내지 15번일 수 있으며, 멀티 코어의 이기종 운영체제에서, 현재의 리소스 할당 방식과 최대한 호환되는 것을 실현하기 위해, 8번 내지 15번(총 8개의 중단)를 사용하여 코어 간 중단 벡터 테이블을 표시한다. 벡터 테이블의 가능한 할당 방안은 표 1에 도시된 바와 같다.

표 1

순서	중단 번호	트리거 소스	응답 소스	의미
1	8	RTOS 코어	Linux 코어	RTOS 시스템에서 CPU 코어에 대한 능동적인 해제
2	9	Linux 코어	RTOS 코어	Linux 시스템에서 CPU 코어에 대한 연결 관리 요청
3	10	RTOS 코어	Linux 코어	RTOS 시스템에서 CPU 코어에 대한 수동적인 해제
4	11	RTOS 코어	Linux 코어	RTOS 시스템에서 CPU 코어에 대한 해제 거부
5	12	RTOS 코어	Linux 코어	RTOS에서 CPU 코어에 대한 선점

[0260] 여기서, 상기 제1 중단 번호는 중단 번호 12에 대응되고, 상기 제2 중단 번호는 중단 번호 8에 대응되며, 상기 제3 중단 번호는 중단 번호 9에 대응되고, 상기 제4 중단 번호는 중단 번호 10에 대응되며, 상기 제5 중단 번호는 중단 번호 11에 대응된다.

[0261] 표 1에 도시된 바와 같이, 능동적인 해제는, RTOS 시스템에 스케줄링할 서비스가 없을 경우(즉, 유향 상태), 중단 번호가 8인 SGI 중단을 Linux 시스템에 발신하는 것을 가리키며, 이때, RTOS의 코어 리소스는 Linux 시스템에 의해 연결 관리되고, RTOS 시스템은 휴면 상태에 진입한다. 수동적인 해제는, Linux 시스템 서비스의 로드가 급증할 경우, 중단 번호가 9인 SGI 중단을 RTOS 시스템에 발신하는 것을 가리키며, 이때 RTOS 시스템에서 실행 중인 프로세스는 중단되는 것(RTOS 시스템에서의 프로세스는 모두 우선순위가 있고, 실제적인 상황에 따라 구성될 수 있으며, 우선순위가 중단 9보다 높을 경우, 중단할 수 없고, 우선순위가 중단 9보다 낮을 경우 중단할 수 있음)을 허용하고, RTOS 시스템에서는 중단 번호가 10인 SGI 중단을 Linux 시스템에 발신하여, Linux 시스템에서 스케줄링하는데 사용되도록, 그에 점유된 CPU 코어 리소스를 수동으로 해제하여 제공하며, 이후 RTOS 시스템은 휴면 상태에 진입하고, 이때 RTOS 시스템이 중단되는 것을 허용하지 않을 경우, 이는 중단 번호가 11인 SGI 중단을 Linux 시스템에 발신하고, 이를 Linux 시스템에 알리며, 이때 RTOS의 코어 리소스는 해제될 수 없고, 이때 Linux 시스템은 현재의 실행 전략에 따라 계속 실행되며, 변하지 않는다.

[0262] 설명해야 할 것은, 코어 간 통신의 벡터 테이블은 유일한 것이 아니며, 상기 표 1에서 한정된 코어 간 통신 벡터 테이블에 한정되지 않는다.

[0263] 선택적인 일 실시예에서는, 코어 간 통신의 방법을 제공한다. 상기 방법은 하기와 같은 단계 1, 단계 2, 단계 3을 포함한다.

[0264] 단계 1에서, 제1 운영체제에서 타깃 데이터(서비스 데이터일 수 있음)를 프로세서 메모리에서의 타깃 가상 채널(저장 공간일 수 있음)에 발신한다.

[0265] 선택적으로, 타깃 데이터는 발신될 데이터이고, 타깃 가상 채널은 메모리에서의 한 세그먼트의 유향 저장 공간이며, 제1 운영체제에서 타깃 데이터를 프로세서 메모리에서의 타깃 가상 채널에 발신하는 것은 제1 운영체제의 CPU 코어에서 발신될 데이터를 타깃 가상 채널에 입력하는 것을 가리킨다.

[0266] 단계 2에서, 제1 운영체제에서 제2 운영체제로 중단 알림 메시지(상기 코어 간 중단 요청일 수 있음)를 발신한다. 선택적으로, 제1 운영체제의 CPU 코어에서 제2 운영체제의 CPU 코어로 중단 알림 메시지를 발신하고, 중단 알림 메시지에는 타깃 가상 채널의 주소를 포함할 수 있으며, 이는 제2 운영체제에 알려 타깃 가상 채널로부터 타깃 데이터를 획득하도록 하는데 사용되며, 중단 알림 메시지는 소프트웨어에 의해 트리거되는 것일 수도 있고, 하드웨어에 의해 트리거되는 것일 수도 있다.

- [0267] 단계 3에서, 제2 운영체제에서 중단 알림 메시지에 응답하여, 메모리에서의 타깃 가상 채널로부터 타깃 데이터를 획득한다. 선택적으로, 제2 운영체제의 CPU 코어에서 중단 알림 메시지에 응답하여, 중단 알림 메시지에서부터 타깃 가상 채널의 주소를 해석하고, 다시 해석된 주소에 따라 메모리에서의 타깃 가상 채널까지 포지셔닝하며, 타깃 가상 채널로부터 타깃 데이터를 획득하여, 제1 운영체제 및 제2 운영체제 사이의 데이터 인터랙션을 실현한다.
- [0268] 상기 단계를 통해, 프로세스에서 실행되는 복수의 운영체제에서 데이터를 서로 전송해야 할 경우, 데이터를 발신하는 제1 운영체제에서 타깃 데이터를 프로세서 메모리에서의 타깃 가상 채널에 발신하고, 제2 운영체제로 중단 알림 메시지를 발신하며, 데이터를 수신하는 제2 운영체제에서 중단 알림 메시지에 응답하여 타깃 가상 채널로부터 타깃 데이터를 획득하여, 코어 간 통신 과정에서 리소스가 낭비되고, 운영체제에 대한 의존성이 강한 과제를 해결함으로써, 코어 간 통신 과정에서 리소스에 대한 낭비 및 운영체제에 대한 의존성을 감소하는 효과에 도달한다.
- [0269] 예시적인 일 실시예에서는, 메모리에는 데이터 저장 영역 및 메타 데이터 저장 영역이 포함되며, 데이터 저장 영역은 복수의 저장 유닛으로 구분되고, 저장 유닛 각각은 서비스 데이터를 저장하며, 메타 데이터 저장 영역은 데이터 저장 영역의 각 저장 유닛의 크기 및 점유된 상태를 저장하기 위한 것이다.
- [0270] 선택적으로, 타깃 가상 채널은 데이터 저장 영역의 하나 또는 복수의 저장 유닛으로 구성되며, 메타 데이터 영역은 저장 유닛의 개수와 동일한 저장 슬라이드로 구분될 수 있고, 저장 슬라이드 각각은 하나의 저장 유닛의 크기 및 점유된 상태를 기록하기 위한 것이며, 저장 유닛의 크기는 저장 유닛의 첫번째 주소와 마지막 주소로 표시될 수도 있고, 첫번째 주소와 저장 유닛의 길이로 표시될 수도 있으며, 점유 상태는 점유된 상태와 점유되지 않는 상태를 포함하며, 유휴 플래그의 개수로 표시될 수 있다.
- [0271] 예시적인 일 실시예에서는, 제1 운영체제에서 타깃 데이터를 프로세서 메모리에서의 타깃 가상 채널에 발신하는 단계는, 제1 운영체제에서 메타 데이터 저장 영역에서의 기록을 읽고, 읽은 기록에 따라 데이터 저장 영역에서 유휴 상태이고, 총 공간이 타깃 데이터의 길이 이상인 적어도 하나의 저장 유닛을 결정하여, 타깃 가상 채널을 획득하는 단계; 메타 데이터 저장 영역에서 타깃 가상 채널에 대응되는 적어도 하나의 저장 유닛의 상태를 점유된 상태로 설정하고, 타깃 데이터를 타깃 가상 채널에 저장하는 단계; 를 포함한다.
- [0272] 설명해야 할 것은, 타깃 데이터가 메모리에 연속으로 입력될 수 있는 것을 보장하기 위해, 입력되는 타깃 가상 채널은 유휴이고, 타깃 데이터의 길이 이상인 저장 공간이어야 한다. 메모리는 메타 데이터 저장 영역 및 데이터 저장 영역으로 구분되므로, 메타 데이터 저장 영역에 기록된 각 저장 유닛의 점유 상태를 읽고, 그 중에서 유휴 상태이고 데이터 저장 수요를 충족시킬 수 있는 저장 유닛을 찾아낼 수 있다.
- [0273] 예를 들면, 저장 유닛 각각의 크기는 동일하며, 타깃 데이터의 길이가 하나의 저장 공간의 길이보다 클 경우, 타깃 데이터의 길이에 따라 수요되는 저장 유닛의 개수를 결정하고, 그 중에서 유휴 상태이고 연속적이며 개수가 데이터 저장 수요를 충족시키는 복수의 저장 유닛을 찾아, 타깃 가상 채널을 구성한다.
- [0274] 또 예를 들면, 저장 유닛 각각의 크기는 동일하며, 데이터 저장 영역은 사전에 저장 유닛을 조합하여, 크기가 서로 다른 복수의 가상 채널을 획득하며, 가상 채널 각각은 하나 또는 복수의 저장 유닛이 조합되어 형성되고, 메타 데이터 저장 영역에 기록된 각 가상 채널의 점유 상태를 읽을 수 있으며, 그 중에서 유휴 상태이고 길이가 타깃 데이터의 길이보다 큰 가상 채널, 즉 타깃 가상 채널을 찾을 수 있다. 설명해야 할 것은, 시스템 소프트웨어에서 공유 메모리 공간을 요청해야 할 경우, 요청해야 할 데이터의 길이가 가상 채널에 저장된 데이터의 최대 길이보다 큰지 여부를 판단하고, 가상 채널에 저장된 데이터의 최대 길이보다 클 경우, 시스템 소프트웨어에서는 발신해야 할 데이터를 복수회로 나누어 발신하여, 매회 발신되는 데이터의 길이가 가상 채널에 저장된 데이터의 최대 길이 이하이도록 보장함으로써, 순리적인 통신 수행을 보장할 수 있다.
- [0275] 예시적인 일 실시예에서는, 제2 운영체제에서 중단 알림 메시지에 응답하여, 메모리에서의 타깃 가상 채널로부터 타깃 데이터를 획득하는 단계는, 제2 운영체제에서 메타 데이터 저장 영역에서의 기록을 읽고, 읽은 기록에 따라 타깃 가상 채널을 결정하는 단계; 타깃 가상 채널에 대응되는 적어도 하나의 저장 유닛으로부터 타깃 데이터를 획득하고, 적어도 하나의 저장 유닛의 상태를 유휴 상태로 설정하는 단계; 를 포함한다.
- [0276] 즉, 제2 운영체제에서 타깃 가상 채널에 대응되는 저장 유닛으로부터 타깃 데이터를 추출한 후, 다른 시스템 또는 태스크에서 타깃 가상 채널을 사용하는데 영향주지 않기 위해, 타깃 가상 채널에 대응되는 저장 유닛의 상태를 유휴 상태로 설정한다.

- [0277] 예시적인 일 실시예에서는, 제1 운영체제에서 타깃 데이터를 프로세서 메모리에서의 타깃 가상 채널에 발신하는 단계는, 제1 운영체제의 구동 계층에서 타깃 데이터를 수신하고, 메모리에서 유휴 상태인 가상 채널을 결정하여, 타깃 가상 채널을 획득하는 단계; 타깃 가상 채널의 상태를 점유된 상태로 설정하고, 타깃 데이터를 타깃 가상 채널에 저장하는 단계; 를 포함한다.
- [0278] 선택적으로, 실시간 운영체제와 비실시간 운영체제는 모두 구동 계층을 갖고, 구동 계층에서 발신될 타깃 데이터를 수신한 후, 인터페이스를 호출하여 메모리에서 타깃 가상 채널은 찾으며, 데이터를 입력하는 과정에서 다른 시스템에서 타깃 가상 채널의 사용을 요청하는 것을 방지하기 위해, 타깃 가상 채널을 찾은 후, 타깃 가상 채널의 상태를 점유된 상태로 설정한 다음, 타깃 데이터를 타깃 가상 채널에 입력한다.
- [0279] 예시적인 일 실시예에서는, 제1 운영체제에 응용 계층이 포함될 경우, 응용 계층에는 인간 컴퓨터간 인터랙션 인터페이스가 설치되고, 제1 운영체제의 구동 계층이 메모리에서 유휴 상태인 가상 채널을 결정하기 전에, 제1 운영체제의 응용 계층에서 인간 컴퓨터간 인터랙션 인터페이스를 통해 사용자에게 의해 입력된 발신될 데이터를 수신하여, 사전 설정된 포맷을 적용하여 발신될 데이터를 캡슐화하여, 타깃 데이터를 획득할 수 있고, 데이터 입력 함수를 호출하여 사전 설정된 통신 인터페이스를 통해 타깃 데이터를 구동 계층에 전송할 수 있으며, 사전 설정된 통신 인터페이스는 구동 계층에 설치된다.
- [0280] 선택적으로, 응용 계층은 발신해야 할 데이터를 사전 설정된 포맷에 따라 채워, 타깃 데이터를 획득한 다음, 시스템의 /dev 경로에서 기기 파일 ipidev을 생성하며, 응용 계층은 구동 계층으로부터 데이터에 대한 읽기/쓰기를 해야 할 경우, 먼저 시스템 자체의 open 함수를 사용하여 기기 파일 /dev/ipidev을 오픈할 수 있고, 다음 시스템 자체의 쓰기 함수를 사용하여 타깃 데이터를 응용 계층으로부터 구동 계층에 발신할 수 있으며, 구동 계층은 데이터를 공유 메모리에서의 타깃 가상 채널에 저장한 다음, 제2 운영체제에서 데이터를 추출하도록 중단 알림을 트리거할 수 있다.
- [0281] 예시적인 일 실시예에서는, 제2 운영체제에서 중단 알림 메시지에 응답하여, 메모리에서의 타깃 가상 채널로부터 타깃 데이터를 획득하는 단계는, 제2 운영체제에서 중단 알림 메시지에 기반하여 중단 처리 함수를 트리거하며, 중단 처리 함수를 통해 메모리로부터 타깃 가상 채널을 결정하고, 타깃 가상 채널로부터 타깃 데이터를 획득하는 단계를 포함한다.
- [0282] 예시적인 일 실시예에서는, 중단 처리 함수를 통해 메모리로부터 타깃 가상 채널을 결정하고, 타깃 가상 채널로부터 타깃 데이터를 획득하는 단계는, 중단 처리 함수를 통해 타깃 태스크를 호출하고, 타깃 태스크에서 메모리로부터 타깃 가상 채널을 결정하며, 타깃 가상 채널로부터 타깃 데이터를 획득하는 단계를 포함한다.
- [0283] 선택적으로, 중단 처리 함수는 태스크 알림을 발신하여 데이터 추출을 담당하는 타깃 태스크를 웨이크업하며, 타깃 태스크는 먼저 인터페이스를 호출하여 공유 메모리에서 타깃 가상 채널을 찾고, 다음 타깃 가상 채널로부터 타깃 데이터를 읽고 데이터를 분석한다.
- [0284] 예시적인 일 실시예에서는, 제2 운영체제에 응용 계층이 포함될 경우, 메모리에는 기능 식별자가 저장되고, 기능 식별자는 타깃 기능을 지시한다. 중단 처리 함수를 통해 메모리로부터 타깃 가상 채널을 결정하고, 타깃 가상 채널로부터 타깃 데이터를 획득하는 단계는, 중단 처리 함수를 통해 메모리로부터 기능 식별자 및 타깃 가상 채널을 결정하고, 타깃 가상 채널의 주소 정보를 기능 식별자에 매칭되는 타깃 응용 프로그램에 발신하되, 타깃 응용 프로그램은 응용 계층에서의 타깃 응용 프로그램인 단계; 타깃 응용 프로그램은 데이터를 호출하여 함수를 읽고, 사전 설정된 통신 인터페이스를 통해 주소 정보를 구동 계층에 전달하며, 구동 계층은 타깃 가상 채널로부터 타깃 데이터를 획득하여, 타깃 데이터를 타깃 응용 프로그램에 전달하되, 사전 설정된 통신 인터페이스는 구동 계층에 설치되고, 타깃 응용 프로그램은 기능 식별자에 매칭되는 처리 함수에 따라 타깃 데이터를 처리하여, 타깃 기능을 실행하는 것인 단계; 를 포함한다.
- [0285] 선택적으로, 제2 운영체제에서 중단 알림 메시지를 수신한 후, 응용 계층은 대응되는 중단 처리 함수를 호출하여 메모리로부터 타깃 가상 채널을 찾아, 타깃 가상 채널의 주소 정보를 획득한 다음, 시스템의 /dev 경로에서 하나의 기기 파일 ipidev을 생성하고, 응용 계층이 구동 계층으로부터 데이터에 대한 읽기/쓰기를 해야 할 경우, 먼저 시스템 자체의 open 함수를 사용하여 기기 파일 /dev/ipidev을 오픈할 수 있고, 다음 시스템 자체의 읽기 함수를 사용하여 타깃 가상 채널에서의 타깃 데이터를 읽을 수 있으며, 즉 구동 계층은 타깃 가상 채널의 주소 정보에 따라 공유 메모리에서 대응되는 타깃 데이터를 찾아, 응용 계층의 타깃 데이터 및 타깃 데이터 길이에 반환한다. 예시적인 일 실시예에서는, 타깃 가상 채널의 상태는 유휴로 설정된다.
- [0286] 설명해야 할 것은, 응용 계층의 서로 다른 응용 프로그램은 타깃 데이터를 이용하여 서로 다른 기능을 실현할

수 있고, 메모리에는 기능 식별자가 저장되며, 응용 프로그램에서 타깃 데이터를 통해 타깃 기능을 실현하도록 지시할 수 있다. 선택적으로, 기능 식별자는 NetFn, Cmd일 수 있고, 시스템이 초기화될 경우, NetFn, Cmd 및 응용 프로그램 PID를 구동 계층에 등록하며, 구동 계층은 수신된 NetFn 및 Cmd에 따라 응용 프로그램인 PID를 찾을 수 있고, PID에 따라 대응되는 응용 프로그램에 데이터를 발신한다.

[0287] 예를 들면, NetFn=1이고 Cmd=1일 경우, 제1 운영체제 및 제2 운영체제 사이에서 “hello word”를 서로 발신하는 것을 나타낸다. 시스템이 시작될 시 하나의 배열을 초기화하며, 배열은 총 3개의 열로서, 제1 열은 NetFn이고, 제2 열은 Cmd이며, 제3 열은 NetFn 및 Cmd에 대응되는 처리 함수로서, xxCmdHandler로 표시된다. 예를 들면, 제2 운영체제에서 제1 운영체제로부터 발신된 메시지를 수신할 경우, 메시지에서부터 NetFn 및 Cmd를 획득하고, NetFn=1이고 Cmd=1인 것으로 판단될 경우, “helloworld”에 대응되는 처리 함수 HelloCmdHandler를 실행하여 대응되는 기능을 완성한다.

[0288] 예시적인 일 실시예에서는, 데이터 저장 영역에는 복수의 메모리 채널이 포함되고, 메모리 채널 각각은 하나 또는 복수의 저장 유닛으로 구성되며, 메타 데이터 저장 영역에는 복수의 기록이 저장되고, 각 기록은 하나의 메모리 채널의 메타 데이터를 기록하기 위한 것이며, 메모리 채널 각각의 메타 데이터는 적어도 메모리 채널의 채널 ID, 메모리 채널의 크기, 메모리 채널의 점유된 상태를 포함한다. 제1 운영체제에서 메타 데이터 저장 영역에서의 기록을 읽고, 읽은 기록에 따라 데이터 저장 영역에서 유향 상태이고, 총 공간이 타깃 데이터의 길이 이상인 적어도 하나의 저장 유닛을 결정하여, 타깃 가상 채널을 획득하는 단계는, 메타 데이터 저장 영역의 기록을 순회하여, 메모리 채널이 유향 상태이고 메모리 채널의 크기가 타깃 데이터의 길이 이상인 것을 지시하는 제1 타깃 기록이 존재하는지 여부를 판단하는 단계; 제1 타깃 기록이 존재할 경우, 제1 타깃 기록에 기록된 채널 ID가 지시하는 메모리 채널을 타깃 가상 채널로 결정하는 단계;를 포함한다.

[0289] 설명해야 할 것은, 데이터 저장 영역을 n개 가상의 메모리 채널로 구분할 수 있으며, 메모리 채널 각각의 크기는 서로 다를 수 있다. 즉, n개의 가상 채널의 크기는 순서에 따라 $20*m$, $21*m$, $22*m$, $23*m$ …… $2n-1*m$ 일 수 있으며, m은 하나의 저장 유닛의 크기이고, 아래와 같은 메타 데이터로서의 구조체를 설정하여 메모리 채널을 관리한다.

```
[0290] typedef struct {
[0291]     uint32_t Flag;
[0292]     uint16_t ChannelId;
[0293]     uint8_t  SrcId;
[0294]     uint8_t  NetFn;
[0295]     uint8_t  Cmd;
[0296]     uint32_t Len;
[0297]     uint32_t ChannelSize;
[0298]     uint8_t  *pData;
[0299]     uint8_t  CheckSum;
[0300] }IpiHeader_T
```

[0301] 여기서, uint32_t Flag은 메모리 채널의 상태를 표시한다. 예를 들면, 0xA5A5A5A5는 이 채널이 비어 있지 않음을 나타내고, 그렇지 않으면 비어있음을 나타낸다. uint16_t ChannelId은 채널 ID를 나타내고, uint8_t SrcId는 소스 CPU ID를 나타내고, 소스 CPU는 메모리 채널로 데이터를 입력하는 CPU를 가리키며, uint8_t NetFn 및 uint8_t Cmd는 기능 파라미터이고, uint32_t Len은 메모리 채널에 저장된 데이터의 길이이고, uint32_t ChannelSize은 메모리 채널의 크기이고, uint8_t *pData는 메모리 채널의 첫번째 주소를 가리키고, uint8_t CheckSum은 체크섬을 가리킨다. 제1 운영체제에서 데이터를 발신해야 할 경우, 발신된 데이터를 체크섬 알고리즘을 통해 체크값을 계산하고, 체크값을 제2 운영체제에 발신하며, 제2 운영체제에서는 데이터 및 체크값을 수신할 경우, 수신된 데이터에 따라 동일한 체크섬 및 알고리즘을 통해 체크값을 계산하고, 계산된 체크값 및 수신된 체크값을 비교하여, 일치할 경우, 수신된 데이터가 유효함을 설명하고, 일치하지 않을 경우, 수신된 데이터가 유효하지 않음을 설명한다. 가상 메모리 채널 각각에는 하나의 구조체 기록이 대응된다. 이 구조체 기록은

채널 ID가 채증하는 방식에 따라 순서대로 공유 메모리의 시작 위치에 저장되고, 상기 구조체 기록은 시스템 전원이 켜진 후 초기화되며, 초기화 Flag이 0이면 이 채널이 비어있음을 나타낸다. 초기화 ChannelId는 순서대로 0, 1, 2 ... n-1이고, 초기화 ChannelSize는 대응되는 가상 메모리 채널의 크기이며, 초기화 pData는 대응되는 가상 메모리 채널의 첫번째 주소를 가리킨다.

[0302] 예시적인 일 실시예에서는, 제1 운영체제에서 타깃 가상 채널을 결정할 경우, 발신될 타깃 데이터의 크기에 따라 인터페이스 GetEmptyChannel를 사용하여 모든 메모리 채널에서, 채널 구조체 IpiHeader에서의 유휴 플래그 Flag가 0xA5A5A5A5(즉, 채널이 유휴 상태임)가 아니고, 또한 채널 구조체 IpiHeader에서의 채널의 크기 ChannelSize가 타깃 데이터의 크기(즉, 메모리의 크기가 타깃 데이터의 저장 수요를 충족시킬 수 있음) 이상인 조건을 충족시키는 가상 채널을 찾는다. 상기 조건을 충족시키는 타깃 가상 채널을 찾은 후, 상기 채널의 상태를 비어있지 않음으로 설정한다. 즉 채널 구조체 IpiHeader에서의 유휴 플래그 Flag를 0xA5A5A5A5로 설정한 다음, 타깃 데이터를 타깃 가상 채널에 복사한다.

[0303] 예시적인 일 실시예에서는, 메모리 채널이 점유될 경우, 메모리 채널의 메타 데이터는 타깃 데이터의 소스 CPU 코어의 ID 및 타깃 데이터의 목적 CPU 코어의 ID를 더 포함한다. 제2 운영체제에서 메타 데이터 저장 영역에서의 기록을 읽고, 읽은 기록에 따라 타깃 가상 채널을 결정하는 단계는, 메타 데이터 저장 영역에 저장된 기록을 순회하여, 제2 타깃 기록이 존재하는지 여부를 판단하되, 제2 타깃 기록은 메모리 채널이 점유된 상태이고, 목적 CPU 코어의 ID는 제2 운영체제의 CPU 코어의 ID이며, 소스 CPU 코어의 ID는 제2 운영체제의 CPU 코어의 ID가 아닌 것을 지시하는 것인 단계; 제2 운영체제 기록이 존재할 경우, 제2 타깃 기록에 기록된 채널 ID는 메모리 채널 ID에 의해 지시되는 메모리 채널을 타깃 가상 채널로 결정하는 단계; 를 포함한다.

[0304] 즉, 타깃 가상 채널은 모든 채널에서 하기와 같은 3개의 조건을 충족시키는 가상 채널이다. 조건 1은, 채널 구조체 IpiHeader에서의 유휴 플래그 Flag가 0xA5A5A5A5인 것이고(즉, 채널이 점유된 상태임을 지시함); 조건 2는, 채널 구조체에서의 TargetId가 현재 CPU의 ID인 것이며(즉, 타깃 데이터의 목적 CPU가 제2 운영체제의 CPU인 것을 지시함), 조건 3은, 채널 구조체에서의 TargetId가 SrcId가 아닌 것이다(즉, 타깃 데이터가 제2 운영체제의 CPU에서 발신된 것이 아님을 지시함).

[0305] 설명해야 할 것은, 1비트를 사용하여 유휴 플래그 Flag를 나타낼 경우, 0은 채널이 비어있음을 나타내고, 1은 채널이 비어있지 않음을 나타낸다. Flag가 기준에는 0이지만, 1로 돌변할 경우, 시스템에서는 Flag를 읽은 후 채널이 비어있지 않는 것으로 간주하여, 통신 예외가 발생한다. 본 실시예에서는, 유휴 플래그 Flag를, 예를 들면, 0xA5A5A5A5와 같은 멀티 비트의 특수 문자로 설정한다. 멀티 비트가 특수 문자로 동시에 돌변할 확률은 1비트가 돌변할 확률보다 작으므로, 저장 매체 비트의 돌변으로 Flag의 값에 영향 주는 것을 방지하여, 통신의 보안을 향상시킬 수 있다.

[0306] 예시적인 일 실시예에서는, 메타 데이터 저장 영역에는 상태 매핑 테이블이 저장된다. 상태 매핑 테이블에는 복수의 기록이 있으며, 기록 각각은 하나의 저장 유닛의 점유된 상태를 기록하기 위한 것이다. 제1 운영체제에서 메타 데이터 저장 영역에서의 기록을 읽고, 읽은 기록에 따라 데이터 저장 영역에서 유휴 상태이고, 총 공간이 타깃 데이터의 길이 이상인 적어도 하나의 저장 유닛을 결정하여, 타깃 가상 채널을 획득하는 단계는, 타깃 데이터에 의해 점유될 저장 유닛의 사전 설정된 개수를 결정하는 단계; 기록 각각에 대해 상태 매핑 테이블의 초기 위치로부터 순서대로 스캔하는 단계; 연속되는 사전 설정된 개수의 타깃 기록이 스캔될 경우, 사전 설정된 개수의 타깃 기록에 의해 지시되는 연속 저장 유닛을 결정하되, 타깃 기록은 저장 유닛이 유휴 상태인 것을 표시하는 것인 단계; 연속 저장 유닛을 타깃 가상 채널로 결정하는 단계; 를 포함한다.

[0307] 설명해야 할 것은, 데이터 저장 및 추출의 편의를 위해, 운영체제에서 서비스 데이터를 전달할 시 메모리에서 연속되는 저장 유닛을 점유해야 하므로, 먼저 메모리 요청 명령어에서의 저장 유닛의 개수를 결정해야 한다. 각 저장 유닛의 메모리 공간은 동일하므로, 소요되는 메모리의 공간 크기를 통해 필요한 연속 저장 유닛의 사전 설정된 개수를 계산하여, numb로 표시할 수 있다.

[0308] 선택적으로, 제1 운영체제는 상태 매핑 테이블에서의 인덱스 위치로부터 기록을 순회한다. 인덱스 위치는 상태 매핑 테이블의 시작 위치일 수 있다. 상태 매핑 테이블의 시작 위치로부터 시작하여, 상태 매핑 테이블의 각 기록을 순서에 따라 조회하여, 유휴 메모리 페이지가 연속으로 기록되 numb 이상인 기록이 존재하는지 여부를 판단한다. 상기 조건에 부합되는 기록이 존재할 경우, 메모리 페이지와의 대응관계를 기록하는 것을 통해, 프로세서에서의 연속 저장 유닛을 결정하고, 상기 연속 저장 유닛을 타깃 가상 채널로 결정하여, 타깃 가상 채널로 데이터를 입력한다.

- [0309] 예시적인 일 실시예에서는, 중단 알림 메시지에는 연속 저장 유닛의 첫번째 주소 및 사전 설정된 개수가 포함된다. 제2 운영체제에서 메타 데이터 저장 영역에서의 기록을 읽고, 읽은 기록에 따라 타깃 가상 채널을 결정하는 단계는, 기록 각각에 대해 상태 매핑 테이블의 초기 위치로부터 순서대로 스캔하는 단계; 연속 저장 유닛이 기록된 첫번째 주소가 스캔될 경우, 스캔된 주소에 의해 지시되는 저장 유닛 및 사전 설정된 개수에서 1을 뺀셈한 연속 저장 유닛을 타깃 가상 채널로 결정하는 단계; 를 포함한다.
- [0310] 선택적으로, 연속 저장 유닛은 개수가 numb인 연속 저장 유닛을 가리킨다. 상태 매핑 테이블에서의 기록 각각에는 저장 유닛에 대응되는 첫번째 주소도 기록된다. 제2 운영체제에서는 매핑 테이블에서 개수가 numb인 연속 저장 유닛의 첫번째 주소의 기록이 스캔될 경우, 타깃 가상 채널의 첫번째 주소가 스캔된 것을 설명한다. 첫번째 주소에 의해 지시되는 저장 유닛 및 상기 저장 유닛 이후의 numb-1 개의 연속 저장 유닛으로 타깃 가상 채널을 구성하고, 제2 운영체제에서는 타깃 가상 채널에서 데이터를 획득하여, 제1 운영체제와의 데이터 인터랙션을 완성한다.
- [0311] 예시적인 일 실시예에서는, 카운터를 통해 스캔된 연속되는 타깃 기록을 기록하고, 기록 각각에 대해 저장 유닛의 개수에 따라 상태 매핑 테이블의 초기 위치로부터 순서대로 스캔하는 과정에서, 현재 타깃 기록이 스캔될 경우, 카운터에 1을 덧셈하도록 제어하고, 현재 비타깃 기록이 스캔될 경우, 카운터가 리셋되도록 제어한다.
- [0312] 선택적으로, 카운터의 개수와 필요한 저장 유닛의 개수의 크기 관계를 이용하여 연속되는 사전 설정된 개수의 타깃 기록이 존재하는지 여부, 즉 사전 설정된 개수의 연속 저장 유닛이 존재하는지 여부를 판단한다. 선택적으로, 카운터의 카운트를 cnt로 표시하며, 스캔된 하나의 저장 유닛이 비어있을 경우, cnt에 1을 덧셈하는 동작을 수행하고, 스캔된 저장 유닛이 비어있지 않을 경우, 누적 연속되는 유향 상태의 저장 유닛의 개수 cnt를 리셋하며, 계속 상기 저장 유닛의 다음 주소에서부터 시작하여 연속되는 유향 상태의 저장 유닛을 찾는다. Cntr가 numb일 경우, 메모리의 수요를 충족시키는 연속되는 유향 상태의 저장 유닛을 찾은 것을 나타낸다. 전반 상태 매핑 테이블을 스캔 완료 후, Cntr가 numb 이상인 경우, 이번 메모리에 대한 동적 요청이 실패되고, 사전 설정된 개수의 연속 저장 유닛이 존재하지 않음을 나타낸다.
- [0313] 예시적인 일 실시예에서는, 제1 운영체제에서 메타 데이터 저장 영역에서의 기록을 읽고, 읽은 기록에 따라 데이터 저장 영역에서 유향 상태이고, 총 공간이 타깃 데이터의 길이 이상인 적어도 하나의 저장 유닛을 결정하여, 타깃 가상 채널을 획득하는 단계 이전에, 상기 방법은 제1 운영체제에서 메모리 요청 명령어를 발신하고, 프로세서의 메모리에 대해 잠금 동작을 실행하되, 메모리 요청 명령어는 프로세서의 메모리에 대한 사용을 요청하기 위한 것인 단계; 메모리에 대한 잠금을 성공할 경우, 상태 매핑 테이블에서의 기록을 읽는 단계; 를 더 포함한다.
- [0314] 선택적으로, 메모리 요청 명령어는 프로세서에서 실행되는 운영체제로부터 발신되는 프로세서의 메모리에 대한 사용을 요청하는 명령어이다. 설명해야 할 것은, 복수의 운영체제에서 프로세서의 메모리에 대한 사용을 동시에 요청할 시 요청 충돌이 발생하는 것을 방지하기 위해, 운영체제에서 메모리 요청 명령어를 발신할 시, 먼저 프로세서의 메모리에 대해 잠금 동작을 실행하고, 잠금 성공 후에야 메모리에 대한 사용을 요청할 수 있다. 잠금 동작은 메모리 요청의 배타 동작을 가리키며, 현재 운영체제에서 잠금 성공 후, 잠금 해제하지 않을 경우, 다른 서버는 프로세서 메모리에 대한 사용을 요청할 권한이 없다.
- [0315] 예시적인 일 실시예에서는, 프로세서의 메모리에 대해 잠금 동작을 실행하는 단계는, 메모리가 현재 잠금된 상태인지 여부를 판단하되, 잠금된 상태는 메모리가 사용이 요청된 상태인 것을 표시하는 것인 단계; 메모리가 현재 잠금된 상태가 아닐 경우, 메모리에 대해 잠금 동작을 실행하는 단계; 메모리가 현재 잠금된 상태일 경우, 메모리의 잠금 실패가 결정되면, 메모리 잠금이 성공할 때까지 또는 잠금 요청의 회수가 사전 설정된 회수보다 클 때까지, 사전 설정된 시간 후 프로세서의 메모리에 대한 잠금을 다시 요청하는 단계; 를 포함한다.
- [0316] 프로세서가 실행되기 전에, 프로세서에서의 메타 데이터 저장 영역 및 데이터 저장 영역에 대해 초기화 동작을 수행해야 한다. 선택적으로, 메타 데이터 저장 영역에서의 상태 매핑 테이블에 저장된 기록을 초기화하고, 메모리 관리 정보를 초기화한다.
- [0317] 메모리에 대한 요청 동작을 수행하기 전에, 메모리 관리 정보에 대해 하기와 같이 구성한다.
- [0318] typedef struct {
- [0319] uint32_t MemReady;
- [0320] uint32_t MemLock;

- [0321] }MallocMemInfo_T;
- [0322] 여기서, 구조체 MallocMemInfo_T의 요소 변수 MemLock는 공유 메모리가 초기화 완료되었는지 여부를 나타내고, 변수 MemReady는 0xA5A5A5A5로서, 초기화 동작이 완료되어, 정상적인 동적 요청 및 메모리 해제를 수행할 수 있는 것을 나타낸다. 구조체 MallocMemInfo_T의 요소 변수 MemReady는 잠금되었는지 여부를 표시한다.
- [0323] 선택적으로, 변수 MemLock가 0인 것을 읽을 경우, 이때 메모리를 요청하는 시스템 또는 태스크가 없는 것을 나타낸다. 즉, 메모리는 현재 잠금된 상태가 아니다. 변수 MemLock가 0xA5A5A5A5인 것을 읽을 경우, 메모리를 요청하는 시스템 또는 태스크가 있어, 이번 요청 완료 후 다시 요청해야 하며, 현재 요청은 잠금 실패인 것을 나타낸다.
- [0324] 예시적인 일 실시예에서는, 메모리에 대해 잠금 동작을 수행할 시 잠금 실패가 발생할 경우, 잠금 성공할 때까지, 사전 설정된 시간동안 대기 후 메모리의 잠금을 다시 요청한다. 예를 들면, 사전 설정된 시간은 100마이크로초일 수 있다.
- [0325] 예시적인 일 실시예에서는, 잠금 요청이 실패되고, 반복 요청 회수가 사전 설정된 회수를 초과하여, 현재 시간 동안 프로세서에서의 메모리가 할당될 수 없는 상태일 경우, 요청 동작은 정지된다. 예를 들면, 사전 설정된 회수는 3회일 수 있으며, 잠금 요청 회수가 3회보다 클 경우, 요청을 발신하는 운영체제에 현재 메모리를 사용할 수 없다는 메시지를 반환할 수 있다.
- [0326] 선택적으로, 프로세서의 메모리 공간에는 제1 운영체제에서 사용될 수 있는 타깃 가상 채널이 존재하고, 제1 운영체제에서는 전송해야 할 타깃 데이터를 대응되는 타깃 가상 채널에 저장한다. 예시적인 일 실시예에서는, 제1 운영체제의 데이터 입력 상황에 따라 프로세서의 메모리 공간의 점유 상태를 업데이트한다. 즉, 타깃 메모리 공간은 점유되지 않은 상태에서 점유된 상태로 변경되고, 동시에 다른 시스템 또는 태스크에서 메모리를 요청할 수 있도록 하기 위해, 메모리의 잠금을 해제한다.
- [0327] 예시적인 일 실시예에서는, 상기 방법은 연속되는 사전 설정된 개수의 타깃 기록이 스캔되지 않을 경우, 메모리에 대한 잠금을 해제하는 단계를 더 포함한다.
- [0328] 선택적으로, 상태 매핑 테이블에서의 기록을 스캔한 후, 사전 설정된 개수의 연속되는 유효 상태의 저장 유닛이 검출되지 않을 경우, 프로세서의 메모리에는 제1 운영체제에서 사용할 수 있는 충족한 공간의 메모리 페이지가 없는 것을 나타내므로, 이번 메모리에 대한 동적 요청은 실패되고, 메모리에 대한 잠금은 해제된다.
- [0329] 예시적인 일 실시예에서는, 소프트웨어 중단 방식을 통해 제2 운영체제로 중단 알림 메시지를 발신한다. 선택적으로, 소프트웨어 중단 방식을 통해 제2 운영체제로 중단 알림 메시지를 발신하는 단계는 프로세서의 사전 설정 레지스터에 중단 번호 및 제2 운영체제의 CPU 코어의 ID를 입력하고, 중단 번호 및 제2 운영체제의 CPU 코어의 ID에 기반하여 중단 알림 메시지를 생성한다.
- [0330] 선택적으로, 소프트웨어 중단은 소프트웨어에 의해 생성되는 중단이고, 소프트웨어는 자체의 CPU 코어에 중단을 발신할 수도 있고, 다른 CPU 코어에 중단을 발신할 수도 있다. 사전 설정 레지스터는 GICD_SGIR 레지스터일 수 있으며, 소프트웨어에서 GICD_SGIR 레지스터로 SGI(Software Generated Interrupts, 소프트웨어 중단) 중단 번호, 목적 CPU ID를 입력하여, 하나의 소프트웨어 중단을 생성할 수 있다. SGI 중단 번호는 코어 간 통신을 위해 예비되는 소프트웨어 중단 번호이다.
- [0331] 선택적으로, 하드웨어 중단은 하드웨어 기기를 통해 생성되는 중단으로서, 사유 주변기기 중단일 수도 있고, 공유 주변기기 중단일 수도 있다. 설명해야 할 것은, 하드웨어 중단은 CPU 외부의 하드웨어에 의해 도입되는 중단으로서, 랜덤성을 갖고, 소프트웨어 중단은 CPU에서 실행되는 소프트웨어에서 중단 명령어를 실행하여 도입되는 중단으로서, 사전에 설정되는 것이다. 본 실시예에서는 중단 알림 메시지를 생성하는 방식에 대해 한정하지 않는다.
- [0332] 선택적인 일 실시예에서는, 메모리를 공유하는 방법을 제공한다. 상기 방법은 하기와 같은 단계 1, 단계 2 및 단계 3을 포함한다.
- [0333] 단계 1에서, 메모리 요청 명령어를 수신하며, 프로세서의 메모리에 대해 잠금 동작을 실행한다. 여기서, 메모리 요청 명령어는 프로세서의 메모리에 대한 사용을 요청하기 위한 것이다.
- [0334] 선택적으로, 메모리 요청 명령어는 프로세서에서 실행되는 운영체제에 의해 발신되는 프로세서의 메모리에 대한 사용을 요청하는 명령어로서, 이는 제1 운영체제에 의해 발신되는 것일 수 있다. 설명해야 할 것은, 복수의 운

운영체제에서 프로세서의 메모리에 대한 사용을 동시에 요청할 시 요청 충돌이 발생하는 것을 방지하기 위해, 운영체제에서 메모리 요청 명령어를 발신할 시, 먼저 프로세서의 메모리에 대해 잠금 동작을 실행하고, 잠금 성공 후에야 메모리에 대한 사용을 요청할 수 있다. 잠금 동작은 메모리 요청의 배타 동작을 가리키며, 현재 운영체제에서 잠금 성공 후, 잠금 해제하지 않을 경우, 다른 서버는 프로세서 메모리에 대한 사용을 요청할 권한이 없다.

- [0335] 본 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 프로세서의 메모리에 대해 잠금 동작을 실행하는 단계 이전에, 이 방법은 메모리가 현재 잠금된 상태인지 여부를 판단하되, 잠금된 상태는 메모리가 사용이 요청된 상태인 것을 표시하는 것인 단계; 메모리가 현재 잠금된 상태가 아닐 경우, 메모리에 대해 잠금 동작을 실행하는 단계; 를 더 포함한다.
- [0336] 선택적으로, 복수의 시스템 또는 복수의 태스크에서 메모리에 대한 사용을 동시에 요청할 시 요청 충돌이 발생하므로, 프로세서의 메모리는 동일한 시간대에 단지 하나의 시스템 또는 태스크에 의해 잠금된다. 따라서, 현재 메모리가 잠금된 상태가 아닌 것이 검출될 경우, 현재 운영체제만 메모리에 대해 잠금 동작을 실행할 수 있다.
- [0337] 선택적으로, 메모리에 저장된 사전 설정된 변수가 사전 설정된 값인지 여부를 판단하여, 메모리가 잠금된 상태인지 여부를 판단한다. 사전 설정된 변수가 사전 설정된 파라미터가 아닐 경우, 메모리는 잠금된 상태가 아니므로, 메모리 공간을 요청하고 있는 다른 시스템 또는 태스크가 없는 것을 나타내어, 잠금 성공이고; 반대로, 사전 설정된 변수가 사전 설정된 파라미터일 경우, 현재 시각 내에 메모리가 잠금 상태이므로, 메모리 공간을 요청하고 있는 상기 운영체제 이외의 다른 시스템 또는 태스크가 존재하는 것을 나타내어, 잠금 실패이다.
- [0338] 상기 메모리를 공유하는 방법에서, 메모리가 현재 잠금된 상태인지 여부를 판단하는 단계 이후, 상기 방법은 메모리가 현재 잠금된 상태일 경우, 메모리에 대한 잠금이 실패인 것을 결정하는 단계; 메모리에 대한 잠금이 실패일 경우, 메모리 잠금이 성공할 때까지 또는 잠금 요청의 회수가 사전 설정된 회수보다 클 때까지, 사전 설정된 시간 후 프로세서의 메모리에 대한 잠금을 다시 요청하는 단계; 를 더 포함한다.
- [0339] 선택적으로, 메모리에 대해 잠금 동작을 수행할 시 잠금 실패가 발생할 경우, 잠금 성공할 때까지, 사전 설정된 시간동안 대기 후 메모리의 잠금을 다시 요청한다. 예를 들면, 사전 설정된 시간은 100마이크로초일 수 있다.
- [0340] 예시적인 일 실시예에서는, 잠금 요청이 실패되고, 반복 요청 회수가 사전 설정된 회수를 초과하여, 현재 시간 동안 프로세서에서의 메모리가 할당될 수 없는 상태일 경우, 요청 동작은 정지된다. 예를 들면, 사전 설정된 회수는 3회일 수 있으며, 잠금 요청 회수가 3회보다 클 경우, 요청을 발신하는 운영체제에 현재 메모리를 사용할 수 없다는 메시지를 반환할 수 있다.
- [0341] 단계 2에서, 메모리에 대한 잠금을 성공할 경우, 메모리의 점유된 상태를 읽고, 메모리의 점유된 상태에 따라 메모리에 유향한 타깃 메모리 공간이 존재하는지 여부를 판단하되, 타깃 메모리 공간의 크기는 메모리 요청 명령어에 의해 요청되는 메모리의 크기 이상이다.
- [0342] 잠금 요청을 성공한 후, 운영체제에서는 프로세서에서의 메모리를 요청한다. 선택적으로, 운영체제에 의해 발신되는 메모리 요청 명령어에 따라, 메모리의 점유된 상태를 기록하기 위한 정보를 스캔하여, 타깃 메모리 공간이 존재하는지 여부를 판단한다. 즉, 프로세서에 점유되지 않은 상태의 연속되는 메모리의 사용 수요를 충족시킬 수 있는 메모리 공간이 존재하는지 여부를 판단한다. 메모리 사용 수요를 충족시키는 것은 메모리 공간의 크기가 운영체제에 의해 요청되는 메모리의 크기 이상인 것을 가리킨다. 예시적인 일 실시예에서는, 점유되지 않은 상태의 연속되는 메모리 공간의 크기가 운영체제에 의해 요청되는 메모리의 크기 이상인지 여부를 판단하여, 판단 결과를 획득한다.
- [0343] 설명해야 할 것은, 메모리를 요청할 시 연속되지 않은 메모리 공간을 사용할 수도 있으며, 하나의 제일 작은 것이 아닌 메모리 블록의 뒤에 하나의 포인터를 추가하여, 다음 요청에서 획득되는 최소 메모리 블록을 지향할 수 있으며, 동시에, 데이터를 읽기/쓰기할 시, 저장 주소 및 포인터에 따라 데이터 블록 전반을 걸친 데이터 읽기/쓰기를 실현할 수 있다. 본 실시예에서는 타깃 메모리 공간의 형태에 대해 한정하지 않는다.
- [0344] 단계 3에서, 메모리에 타깃 메모리 공간이 존재할 경우, 타깃 메모리 공간의 주소 정보를 메모리 요청 명령어의 발신측에 피드백하여, 메모리의 점유 상태를 업데이트하며, 메모리에 대한 잠금을 해제한다. 여기서, 발신측은 메모리 요청 명령어를 발신하는 운영체제(예를 들면, 제1 운영체제)를 가리킨다. 설명해야 할 것은, 운영체제에서 코어 간 통신을 수행할 시, 공유 메모리를 사용하여 데이터를 송수신하고, 데이터를 송수신하는 과정에서 요청된 메모리에서 반환된 주소를 사용하여 데이터의 접근을 수행하므로, 요청된 메모리 공간의 주소 정보를 결정

해야 한다.

- [0345] 선택적으로, 프로세서의 메모리 공간에 운영체제에서 사용될 수 있는 타깃 메모리 공간(상기 판단 결과에서 지시된 것일 수 있음)이 존재할 경우, 타깃 연속 공간의 주소 정보를 상기 운영체제에 발신하고, 운영체제에서는 주소 정보에 따라 전송해야 할 데이터를 대응되는 메모리 공간에 저장한다.
- [0346] 예시적인 일 실시예에서는, 운영체제의 데이터 입력 상황에 따라 프로세서의 메모리 공간의 점유 상태를 업데이트한다. 즉, 타깃 메모리 공간은 점유되지 않은 상태에서 점유된 상태로 변경되고, 다른 운영체제에서 프로세서의 메모리 공간에 대한 사용을 요청할 수 있도록, 메모리에 대한 동적 요청 이전의 잠금을 해제한다.
- [0347] 상기 단계를 통해, 복수의 커널 간 공유 메모리의 사용효율이 낮고, 원활성이 낮으며, 운영체제에 지나치게 의존하는 과제를 해결하여, 메모리 공유의 원활성 및 사용효율을 향상시키고, 운영체제에 대한 의존도를 줄이는 효과에 도달할 수 있다.
- [0348] 메모리를 공유하는 방법에서, 메모리에는 메타 데이터 저장 영역 및 데이터 저장 영역이 포함되며, 데이터 저장 영역은 서비스 데이터를 저장하고, 메타 데이터 저장 영역에는 상태 매핑 테이블이 저장된다. 상태 매핑 테이블은 데이터 저장 영역의 점유된 상태를 기록하기 위한 것이다. 메모리의 점유된 상태를 읽고, 메모리의 점유된 상태에 따라 메모리에 유희한 타깃 메모리 공간이 존재하는지 여부를 판단하는 단계는, 메타 데이터 저장 영역으로부터 상태 매핑 테이블에서의 기록을 읽고, 상태 매핑 테이블에서의 기록에 따라 데이터 저장 영역에 타깃 메모리 공간이 존재하는지 여부를 판단하는 단계를 포함한다.
- [0349] 상태 매핑 테이블에서의 기록을 조회하는 방식을 통해 메모리의 점유된 상태를 조회한다. 선택적으로, 프로세서에 저장된 메타 데이터 저장 영역을 획득하고, 메타 데이터 저장 영역에서의 상태 매핑 테이블을 식별하며, 상태 매핑 테이블에서의 기록을 순회하여, 데이터 저장 영역의 점유된 상태를 읽고, 데이터 저장 영역에 연속되는 유희 상태의 메모리 사용 수요를 충족시키는 메모리 공간이 존재하는지 여부를 판단한다.
- [0350] 본 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 데이터 저장 영역은 복수의 메모리 페이지로 구성되고, 상태 매핑 테이블에는 복수의 기록이 포함되며, 기록 각각은 하나의 메모리 페이지의 점유된 상태를 기록하기 위한 것이다. 메타 데이터 저장 영역으로부터 상태 매핑 테이블에서의 기록을 읽고, 상태 매핑 테이블에서의 기록에 따라 데이터 저장 영역에 타깃 메모리 공간이 존재하는지 여부를 판단하는 단계는, 메모리 요청 명령어에 의해 요청되는 메모리 페이지의 사전 설정된 개수를 결정하는 단계; 기록 각각에 대해 상태 매핑 테이블의 초기 위치로부터 순서대로 스캔하는 단계; 연속되는 사전 설정된 개수의 타깃 기록이 스캔될 경우, 메모리에 타깃 메모리 공간이 존재하는 것으로 결정하되, 타깃 기록은 메모리 페이지가 유희 상태인 것을 나타내는 것인 단계; 를 포함한다.
- [0351] 설명해야 할 것은, 데이터 저장 영역은 동일한 메모리 크기에 따라 복수의 할당 유닛으로 구분되며, 각 할당 유닛은 하나의 메모리 페이지로 표시된다. 예를 들면, 데이터 저장 영역의 메모리 공간이 A바이트이고, 구분된 할당 유닛이 B바이트일 경우, 상기 데이터 저장 영역은 총 A/B개의 메모리 페이지를 포함한다. 상태 매핑 테이블에서의 기록이 바로 메모리 페이지 기록이고, 메모리 페이지 기록 각각은 하나의 메모리 페이지의 점유된 상태를 기록하기 위한 것이며, 상태 매핑 테이블에서의 메모리 페이지 기록 및 데이터 저장 영역에서의 메모리 페이지의 개수는 동일하다.
- [0352] 도 6은 본 출원의 실시예에 따른 메모리를 공유하는 방법에서 상태 매핑 테이블 및 메모리 페이지의 관계 예시도이다. 도 6에 도시된 바와 같이, 데이터 저장 영역이 바로 동적 할당 메모리 블록 영역이고, 메타 데이터 저장 영역은 동적 할당 메모리 매핑 테이블 영역을 포함한다. 여기서, 매핑 테이블 영역은 데이터 저장 영역에서 구분된 메모리 페이지의 개수에 따라 동일한 개수의 기록으로 구분되고, 상기 기록은 메모리 페이지 기록으로 표시되며, 모든 메모리 페이지 기록을 조합하여 상태 매핑 테이블을 형성한다. 상태 매핑 테이블에는 모든 메모리 페이지 기록과 데이터 저장 영역의 모든 메모리 페이지가 일대일 대응되는 관계가 존재하며, 각 메모리 페이지 기록에서 대응되는 메모리 페이지의 할당 상태, 즉 메모리 페이지의 점유되었는지 여부를 나타낸다.
- [0353] 선택적으로, 운영체제에서 협동하는 서비스 데이터는 프로세서에서 연속되는 메모리 페이지를 점유해야 하므로, 먼저 메모리 요청 명령어에서의 메모리 페이지의 사전 설정된 개수를 결정해야 한다. 각 메모리 페이지의 메모리 공간은 동일하므로, 수요되는 메모리의 공간 크기를 통해 필요한 연속 메모리 페이지의 사전 설정된 개수를 계산할 수 있으며, numb로 표시한다.
- [0354] 예시적인 일 실시예에서는, 프로세서의 메타 데이터 저장 영역에서의 상태 매핑 테이블을 획득한 다음, 상태 매핑 테이블에서의 인덱스 위치로부터 메모리 페이지 기록을 순회한다. 인덱스 위치는 상태 매핑 테이블의 시작

위치일 수 있다. 상태 매핑 테이블의 시작 위치로부터 시작하여, 상태 매핑 테이블의 각 메모리 페이지의 기록을 순서에 따라 조회하여, 연속되는 유향 메모리 페이지를 기록하는 numb 이상인 메모리 페이지 기록이 존재하는지 여부를 결정한다. 상기 조건에 부합되는 메모리 페이지 기록이 존재할 경우, 메모리 페이지와의 대응관계를 통해, 프로세서에 타깃 메모리 공간이 존재하는 것으로 결정한다.

[0355] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 기록 각각에 대해 상태 매핑 테이블의 초기 위치로부터 순서대로 스캔하는 단계 이후, 상기 방법은 상태 매핑 테이블에서의 모든 기록을 스캔 완료하고, 연속되는 사전 설정된 개수의 타깃 기록이 존재하지 않을 경우, 메모리에 타깃 저장 공간이 존재하지 않는 것으로 결정하는 단계를 더 포함한다.

[0356] 선택적으로, 상태 매핑 테이블의 초기 위치로부터 시작하여, 상태 매핑 테이블의 메모리 페이지 기록을 조회하여, 연속되는 메모리 페이지 개수가 numb 이상인 공간이 존재하는지 여부를 결정하고, 전반 상태 매핑 테이블을 스캔 완료 후 연속되는 사전 설정된 개수의 유향 메모리 페이지를 여전히 발견하지 못할 경우, 타깃 메모리 공간이 존재하지 않는 것을 나타낸다.

[0357] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 카운터를 통해 스캔된 타깃 기록의 개수를 기록하고, 기록 각각에 대해 상태 매핑 테이블의 초기 위치로부터 순서대로 스캔하는 과정에서, 현재 타깃 기록이 스캔될 경우, 카운터에 1을 덧셈하도록 제어하고, 현재 비타깃 기록이 스캔될 경우, 카운터가 리셋되도록 제어한다. 여기서, 비타깃 기록은 메모리 페이지가 점유된 상태인 것을 지시한다.

[0358] 선택적으로, 카운터의 수치와 필요한 메모리 페이지의 개수의 크기 관계를 이용하여 연속되는 사전 설정된 개수의 타깃 기록이 존재하는지 여부, 즉 타깃 메모리 공간이 존재하는지 여부를 판단한다. 선택적으로, 카운터의 카운트를 cnt로 표시하며, 스캔된 하나의 메모리 페이지가 비어있을 경우, cnt에 1을 덧셈하는 동작을 수행하고, 스캔된 메모리 페이지가 비어있지 않을 경우, 누적 연속되는 유향 상태의 메모리 페이지의 개수 cnt를 리셋하며, 계속 상기 메모리 페이지의 다음 주소에서부터 시작하여 연속되는 유향 상태의 메모리 페이지를 찾는다. Cntr가 numb일 경우, 메모리의 수요를 충족시키는 연속되는 유향 상태의 메모리 페이지를 찾은 것을 나타낸다. 전반 상태 매핑 테이블을 스캔 완료하는 과정에서, Cntr가 numb보다 작을 경우, 이번 메모리에 대한 동적 요청이 실패되고, 타깃 메모리 공간이 존재하지 않음을 나타낸다.

[0359] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 초기 위치가 상태 매핑 테이블에서의 마지막 위치일 경우, 타깃 메모리 공간의 주소 정보를 메모리 요청 명령어의 발신측에 피드백하는 단계는, 연속되는 사전 설정된 개수의 타깃 기록에서 마지막으로 스캔된 타깃 기록을 결정하고, 마지막으로 스캔된 타깃 기록에 의해 지시되는 메모리 페이지의 첫번째 주소를 발신측에 피드백하는 단계를 포함한다.

[0360] 선택적으로, 상태 매핑 테이블을 스캔할 경우, 스캔 방식은 상태 매핑 테이블의 첫번째 위치로부터 스캔하거나 또는 상태 매핑 테이블의 마지막 위치로부터 시작하여 스캔하는 방식을 선택할 수 있다. 스캔 방식이 상태 매핑 테이블의 마지막 위치로부터 스캔하는 방식일 경우, 카운터에 디스플레이된 수치 cnt가 사전 설정된 개수 numb 이상이면, 스캔된 마지막 메모리 페이지에는 대응되는 메모리 페이지의 첫번째 주소가 기록되고, 메모리 페이지 기록에서 이러한 메모리 페이지의 상태를 비어있지 않은 것으로 설정하며, 첫번째 주소를 이번 메모리 요청 명령어의 전반 연속되는 메모리 페이지의 첫번째 주소로 간주한다.

[0361] 예시적인 일 실시예에서는, 상기 주소를 메모리 요청 명령어를 발신하는 운영체제에 피드백하고, 운영체제에서는 주소 정보에 따라 메모리에 대해 데이터 입력 동작을 수행한다.

[0362] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 초기 위치가 상태 매핑 테이블에서의 첫번째 위치일 경우, 타깃 메모리 공간의 주소 정보를 메모리 요청 명령어의 발신측에 피드백하는 단계는, 연속되는 사전 설정된 개수의 타깃 기록에서 첫번째로 스캔된 타깃 기록을 결정하고, 첫번째로 스캔된 타깃 기록에 의해 지시되는 메모리 페이지의 첫번째 주소를 발신측에 피드백하는 단계를 포함한다.

[0363] 선택적으로, 스캔 방식이 상태 매핑 테이블의 첫번째 위치로부터 스캔하는 방식일 경우, 카운터에 디스플레이된 수치 cnt가 사전 설정된 개수 numb 이상이면, 스캔된 첫번째 메모리 페이지에 기록된 주소를 첫번째 주소로 간주하고, 상기 주소를 메모리 요청 명령어를 발신하는 운영체제에 피드백하며, 운영체제에서는 주소 정보에 따라 메모리에 대해 데이터 입력 동작을 수행한다.

[0364] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 기록 각각에 대해 상태 매핑 테이블의 초기 위치로부터 순서대로 스캔하는 과정에서, 사전 설정된 변수를 통해 스캔된 연속되는 타깃 기록에서의 첫번째 타깃

기록을 저장한다.

- [0365] 선택적으로, 사전 설정된 변수는 상태 매핑 테이블에서 초기 위치의 주소 정보를 저장하기 위한 변수를 가리키며, 이를 offset로 표시한다. 하나의 유희하고 연속되는 메모리 페이지가 스캔될 때마다, 카운터에 디스플레이된 수치 cnt에 1을 덧셈하는 동작을 수행하며, 카운터에 디스플레이된 수치 cnt가 사전 설정된 개수 numb 이상인 경우, offset 현재 저장된 주소 정보를 첫번째 타깃 기록의 주소로 간주한다.
- [0366] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 메모리의 점유된 상태를 읽고, 메모리의 점유된 상태에 따라 메모리에 유희한 타깃 메모리 공간이 존재하는지 여부를 판단하는 단계 이후, 상기 방법은 메모리에 유희한 타깃 메모리 공간이 존재하지 않을 경우, 메모리에 대한 잠금을 해제하는 단계를 더 포함한다.
- [0367] 선택적으로, 상태 매핑 테이블에서의 메모리 페이지 기록을 스캔한 후, 사전 설정된 개수의 연속되는 유희한 메모리 페이지가 포함되지 않고, 즉 타깃 메모리 공간이 포함되지 않는 것이 검출될 경우, 프로세서의 메모리에 상기 운영체제에 의해 사용될 충분한 공간의 메모리 페이지가 존재하지 않는 것을 나타낸다. 이번 메모리에 대한 동적 요청은 실패되고, 메모리에 대한 잠금은 해제된다.
- [0368] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 메모리에는 메타 데이터 저장 영역 및 데이터 저장 영역이 포함되며, 데이터 저장 영역은 서비스 데이터를 저장하고, 메타 데이터 저장 영역에는 메모리 관리 정보가 저장된다. 메모리가 현재 잠금된 상태인지 여부를 판단하는 단계는, 메타 데이터 저장 영역에 저장된 메모리 관리 정보를 읽고, 메모리 관리 정보에 사전 설정된 정보가 포함되는지 여부를 판단하고, 사전 설정된 정보는 메모리가 잠금된 상태인 것을 표시하는 것인 단계; 메모리 관리 정보에 사전 설정된 정보가 포함될 경우, 메모리가 현재 잠금된 상태가 아닌 것으로 결정하고, 메모리 관리 정보에 사전 설정된 정보가 포함되지 않을 경우, 메모리가 현재 잠금된 상태인 것으로 결정하는 단계; 를 포함한다.
- [0369] 프로세서의 메모리가 잠금된 상태인지 여부를 판단할 경우, 메타 데이터 저장 영역에서의 메모리 관리 정보를 이용하여 판단해야 한다. 선택적으로, 메타 데이터 저장 영역의 메모리 관리 정보를 획득할 시, 메모리 관리 정보에 사전 설정된 정보가 포함되는지 여부를 판단해야 한다. 여기서, 사전 설정된 정보는 메모리가 잠금된 상태를 인지를 표시하기 위한 것이고, 메모리 관리 정보에 사전 설정된 정보가 포함되지 않을 경우, 현재 메모리는 잠금된 상태가 아니고, 반대일 경우, 잠금된 상태인 것을 나타낸다.
- [0370] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 메모리 관리 정보는 제1 필드 정보 및 제2 필드 정보를 포함한다. 제1 필드 정보는 메모리가 잠금된 상태인지 여부를 설명하기 위한 것이고, 제2 필드 정보는 메모리가 초기화 완료되었는지 여부를 설명하기 위한 것이다. 메모리 요청 명령어를 수신하는 단계 이전에, 이 방법은 데이터 저장 영역에 저장된 제1 필드 정보 및 제2 필드 정보를 초기화하는 단계를 더 포함한다.
- [0371] 임베디드 시스템이 실행되기 전에, 프로세서에서의 메타 데이터 저장 영역 및 데이터 저장 영역을 초기화해야 한다. 선택적으로, 메타 데이터 저장 영역에서의 상태 매핑 테이블에 저장된 메모리 페이지를 초기화하고, 메모리 관리 정보를 초기화한다.
- [0372] 선택적으로, 메모리 관리 정보는 제1 필드 정보 및 제2 필드 정보로 구성되고, 제1 필드 정보는 잠금되었는지를 표시하기 위한 것이고, 제2 필드 정보는 초기화 완료되었는지를 표시하기 위한 것이다.
- [0373] 본 출원의 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 메모리의 점유된 상태를 업데이트 하는 단계는, 상태 매핑 테이블에 기록된 타깃 메모리 공간에 대응되는 메모리 페이지의 상태를 점유된 상태로 변경하는 단계를 포함한다.
- [0374] 선택적으로, 운영체제에서 타깃 메모리 공간을 점유해야 할 경우, 타깃 메모리 공간의 복수의 메모리 페이지의 주소 정보를 식별하여, 메모리 페이지와 메모리 페이지 기록의 대응관계에 따라, 메타 데이터 저장 영역의 상태 매핑 테이블의 메모리 페이지 기록을 업데이트하여, 점유되지 않은 상태에서 점유된 상태로 변경되도록 한다. 진일보로, 운영체제의 데이터 입력 상황에 따라 프로세서의 메모리 공간의 점유 상태를 업데이트한다. 즉, 타깃 메모리 공간을 점유되지 않은 상태에서 점유된 상태로 변경하고, 메모리에 대한 동적 요청 이전의 잠금 동작을 해제한다.
- [0375] 선택적으로, 제1 운영체제의 저장 동작에 응답하여, 타깃 데이터를 타깃 메모리 공간에 저장하고, 연속되는 메모리 공간의 주소 정보를 제2 운영체제에 발신하며, 제2 운영체제에서 주소 정보에 기반하여 발신되는 획득 명령어를 수신하여, 타깃 메모리 저장 공간에 저장된 타깃 데이터를 제2 운영체제에 발신한다.
- [0376] 여기서, 메모리 요청을 성공한 후, 제1 운영체제에서는 전달해야 할 타깃 데이터를 요청된 타깃 메모리 공간에

저장하고, 제1 운영체제와 협동하는 제2 운영체제로 상기 타깃 메모리 공간의 주소 정보를 발신하며, 데이터를 획득하도록 제2 운영체제에 알린다. 선택적으로, 제2 운영체제에서 타깃 메모리 공간의 주소 정보를 수신한 후, 데이터의 획득 명령어를 발신하고, 임베디드 시스템에서 상기 명령어를 수신하여, 타깃 메모리 공간에 저장된 타깃 데이터를 제2 운영체제에 발신한다.

[0377] 상기 단계에서, 제1 운영체제의 메모리 요청 명령어를 수신하고, 프로세서의 메모리에 대해 잠금 동작을 실행하되, 메모리 요청 명령어는 프로세서의 메모리에 대한 사용을 요청하기 위한 것인 단계; 메모리에 대한 잠금을 성공할 경우, 메모리의 점유된 상태를 읽고, 메모리의 점유된 상태에 따라 메모리에 유향한 타깃 메모리 공간이 존재하는지 여부를 판단하되, 타깃 메모리 공간의 크기는 메모리 요청 명령어에 의해 요청되는 메모리의 크기 이상인 것인 단계; 메모리에 타깃 메모리 공간이 존재할 경우, 타깃 메모리 공간의 주소 정보를 메모리 요청 명령어의 발신측에 피드백하여, 메모리의 점유 상태를 업데이트하며, 메모리에 대한 잠금을 해제하는 단계; 제1 운영체제의 저장 동작에 응답하여, 타깃 데이터를 타깃 메모리 공간에 저장하고, 연속되는 메모리 공간의 주소 정보를 제2 운영체제에 발신하는 단계; 제2 운영체제에서 주소 정보에 기반하여 발신되는 획득 명령어를 수신하여, 타깃 메모리 저장 공간에 저장된 타깃 데이터를 제2 운영체제에 발신하는 단계; 를 통해, 복수의 커널 간 공유 메모리의 사용효율이 낮고, 원활성이 낮으며, 운영체제에 지나치게 의존하는 과제를 해결하여, 메모리 공유의 원활성 및 사용효율을 향상시키고, 운영체제에 대한 의존도를 줄이는 효과에 도달할 수 있다.

[0378] 예시적인 일 실시예에서는, 제1 운영체제에서 물리적 주소를 사용하여 데이터에 대한 읽기/쓰기 동작을 수행하고, 제2 운영체제에서 가상 주소를 사용하여 데이터에 대한 읽기/쓰기 동작을 수행할 경우, 제2 운영체제에서는 타깃 메모리 공간의 주소 정보를 가상 주소로 변환하고, 가상 주소를 적용하여 메모리에 액세스하여, 타깃 메모리 공간으로부터 타깃 데이터를 읽는다.

[0379] 코어 간 통신에서 공유 메모리를 사용하여 데이터를 송수신할 시, 동적 요청되는 메모리에 의해 반환된 주소를 사용한다. 하지만 서로 다른 시스템에 의해 사용되는 주소의 시스템은 서로 다를 수 있다. 예를 들면, 실시간 운영체제는 제1 운영체제이고, 비실시간 운영체제는 제2 운영체제이다. 실시간 운영체제에서는 직접 물리적 주소를 사용하여 공유 메모리에 액세스할 수 있으며, 비실시간 운영체제에서는 직접 물리적 주소를 사용하여 공유 메모리에 액세스할 수 없고, 매핑된 가상 주소를 사용해야 한다. 제2 운영체제에서 타깃 메모리 공간의 주소 정보를 수신한 후, 주소 정보 offset를 통해 변환하고, 이를 가상 주소로 매핑하며, 가상 주소에 따라 동작한다. 선택적으로, 비실시간 운영체제에서의 공유 메모리의 가상 주소는 vBase(공유 메모리의 진실한 물리적 주소가 0x96000000이라고 가정함)이고, 실시간 운영체제에서 공유 메모리의 가상 주소는 pBase(즉, 0x96000000임)이다.

[0380] 비실시간 운영체제에서 동적 요청되는 메모리에 의해 반환된 주소도 가상 주소 vData이고, 비실시간 운영체제에서는 $\text{Offset} = \text{vData} - \text{vBase}$ 이다. 데이터의 발신은 비실시간 운영체제로부터 실시간 운영체제로 발신되며, 실시간 운영체제에서 주소 pData를 사용하여 동적 요청되는 공유 메모리 $\text{pData} = \text{pBase} + \text{Offset}$ 를 액세스한다. 실시간 운영체제에서 동적 요청되는 메모리에 의해 반환된 주소는 물리적 주소 pData이고, 실시간 운영체제에서는 $\text{Offset} = \text{pData} - \text{pBase}$ 이다. 데이터의 발신은 실시간 운영체제로부터 비실시간 운영체제로 발신되며, 비실시간 운영체제에서 주소 vData를 사용하여 동적 요청되는 공유 메모리 $\text{vData} = \text{vBase} + \text{Offset}$ 를 액세스한다.

[0381] 예시적인 일 실시예에서는, 메모리에는 메타 데이터 저장 영역 및 데이터 저장 영역이 포함되며, 메타 데이터 저장 영역 및 데이터 저장 영역은 상기 실시예와 유사하며, 여기에서 설명을 생략한다. 선택적으로, 프로세서에 저장된 메타 데이터 저장 영역을 획득하고, 메타 데이터 저장 영역에서의 상태 매핑 테이블을 식별하며, 상태 매핑 테이블에서의 인덱스 위치로부터 시작하여 각 메모리 페이지 기록을 순회하고, 상태 매핑 테이블의 각 메모리 페이지의 기록을 순서에 따라 조회하여, 연속되는 유향 메모리 페이지를 기록하는 사전 설정된 개수 이상인 메모리 페이지 기록이 존재하는지 여부를 결정한다. 상기 조건에 부합되는 메모리 페이지 기록이 존재할 경우, 메모리 페이지 기록과 메모리 페이지의 대응관계를 통해, 프로세서에 타깃 메모리 공간이 존재하는 것으로 결정한다.

[0382] 본 실시예에 의해 제공되는 메모리를 공유하는 방법에서, 이 방법은, 운영체제에서 메모리 요청 명령어를 발신하기 전에, 복수의 운영체제에서 프로세서의 메모리 공간을 동시에 요청할 시 요청 충돌이 발생하는 것을 방지하기 위해, 잠금 동작을 요청하고, 잠금 성공되었는지 여부를 판단해야 하는 단계; 판단 결과가 메모리의 잠금에 대한 동적 요청이 성공한 것을 나타낼 경우, 발신된 메모리 요청 명령어에서 메모리 크기에 따라 할당해야 할 연속 메모리 페이지의 페이지 수를 계산하고, nmemb로 표시하는 단계; 판단 결과가 잠금 요청이 실패한 것을 나타낼 경우, 요청 성공할 때까지, 일정한 시간 동안(100마이크로초일 수 있음) 대기한 후 다시 요청을 발신하며, 잠금에 대한 요청이 실패한 회수가 사전 설정된 회수(사전 설정된 회수는 3회일 수 있음)보다 클 경우, 메

모리 요청을 종료한다.

- [0383] 예시적인 일 실시예에서는, 잠금 요청 성공 후, 프로세서의 메타 데이터 저장 영역을 초기화하고, 상태 매핑 테이블의 마지막 위치를 offset로 표시하며, 메모리 요청 명령어에서의 수요되는 메모리의 크기에 따라 필요한 연속 메모리 페이지의 개수를 계산하고, 메모리 페이지 개수를 nmemb로 표시하며, 메모리 페이지 개수를 기록하기 위한 카운터를 설정하며, cmemb로 표시한다. 다음 프로세서에서의 메타 데이터 저장 영역의 상태 매핑 테이블을 획득하며, 상태 매핑 테이블의 offset 위치로부터 시작하여 전반 상태 매핑 테이블을 스캔하고, 상태 매핑 테이블에 저장된 메모리 페이지 기록과 데이터 저장 영역에서 메모리 페이지의 대응관계를 통해, 연속되는 비어있는 메모리 페이지를 찾는다. 스캔된 현재의 메모리 페이지가 점유된 상태일 경우, offset=offset-cmemb하고, 다음 카운터에 누적 연속되는 비어있는 메모리 페이지의 데이터를 cmemb 리셋하며, 계속하여 새로운 offset 위치로부터 다시 연속되는 비어있는 메모리 페이지를 찾는다. 스캔된 메모리 페이지가 비어있음, 즉 유휴 상태일 경우, 카운터의 수치 cmemb에 1을 덧셈하고, offset=offset+1하며, cmemb이 nmemb와 같을 때까지, 계속 다음 메모리 페이지를 판단한다. 즉, 카운터 데이터와 수요되는 메모리의 공간 크기가 동일할 경우, 요구사항을 충족시키는 연속 메모리 페이지가 스캔된 것을 나타낸다.
- [0384] 예시적인 일 실시예에서는, 요구사항에 부합되는 메모리 페이지를 대응되는 상태 매핑 테이블에서는 점용된 상태로 라벨링되고, 마지막으로 찾은 메모리 페이지의 첫번째 주소는 동적 요청되는 전반 연속되는 메모리 페이지의 첫번째 주소로서, 동적 요청되는 메모리의 잠금을 해제하며, 이번 메모리에 대한 동적 요청은 성공이다.
- [0385] 전반 상태 매핑 테이블을 스캔하는 과정에서, offset의 값이 0보다 작을 경우, 운영체제에 의해 사용되는 요구사항에 부합되는 메모리 페이지가 없는 것을 나타내며, 동적 요청되는 메모리의 잠금은 해제되고, 메모리에 대한 동적 요청은 실패이다.
- [0386] 또한, 공간을 동적 요청한 후 공간이 부족할 경우, 크기에 대해 동적 조정할 수도 있고, 업데이트 후의 메모리 요청 명령어를 다시 발신할 수 있으며, 메모리에 대해 잠금 동작을 수행할 수 있다. 잠금 성공할 경우, 업데이트 후의 메모리 요청 명령어로 요청해야 되는 메모리 공간이 증가한다면, 요청된 타깃 연속 메모리 이후에 수요되는 저장 공간이 존재하는지 여부를 판단하며, 존재할 경우, 요청 성공이고, 업데이트 후의 메모리 요청 명령어로 요청해야 되는 메모리 공간이 감소한다면, 일부분의 메모리 공간을 해제한다.
- [0387] 본 실시예에서는, 복수의 저장 영역으로 구분되고, 인덱스 위치를 이용하여 실제 필요한 공간 크기에 따라 동적 요청하며, 사용 완료 후 해제하고, 공간에 대해 동적 요청 후 공간이 부족할 경우 크기에 대해 동적 조정할 수도 있어, 공유 메모리의 원활성 및 사용효율을 향상시키는 효과에 도달할 수 있다.
- [0388] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S221 및 S222 중의 적어도 한 단계를 더 포함한다.
- [0389] S221에서, 제1 운영체제에 의해 선점된 제2 운영체제의 처리 리소스를 제1 운영체제의 사용 가능한 리소스풀에 병합한다.
- [0390] S222에서, 제2 운영체제에 의해 선점된 제1 운영체제의 처리 리소스 또는 제1 운영체제에서 제2 운영체제에 해제한 처리 리소스를 제2 운영체제의 사용 가능한 리소스풀에 병합한다.
- [0391] 본 실시예에서는, 처리 리소스의 스케줄링은 대응되는 처리 리소스를 대응되는 운영체제의 사용 가능한 리소스풀에 병합하는 것을 통해 실현되는 것일 수 있다. 대응되게, 제1 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 장면에 있어서, 리소스 선점을 성공할 경우, 제1 운영체제에 의해 선점된 제2 운영체제의 처리 리소스를 제1 운영체제의 사용 가능한 리소스풀에 병합할 수 있다. 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하고, 제1 운영체제에서 제2 운영체제에 처리 리소스를 능동적으로 해제하는 장면에 있어서, 리소스 선점을 성공하거나 또는 리소스 해제를 성공할 경우, 제2 운영체제에 의해 선점된 제1 운영체제의 처리 리소스 또는 제1 운영체제에 의해 능동적으로 해제된 처리 리소스를 제1 운영체제의 사용 가능한 리소스풀에 병합할 수 있다.
- [0392] 예를 들면, 리소스 적응형 스케줄링 모듈은, 예를 들면, Linux 시스템의 SMP(Symmetrical Multi-Processing, 대칭형 멀티 프로세서) 프레임과 같은 제1 운영체제의 프로세서 프레임 및 제2 운영체제의 프로세서 프레임에 협력하여 프로세서 하드 코어 리소스풀을 구성하여 실현할 수 있다. 여기서, 하나의 컴퓨터에는 한세트의 프로세서(멀티 CPU)가 집중되고, 각 CPU 사이에서는 메모리 서브 시스템 및 버스 구조를 공유하며, SMP 스케줄링은 프로세스를 적합한 CPU에 배치/이전하여, 각 CPU 로드 밸런스를 유지하는 과정이다.
- [0393] Linux 시스템을 예로 들면, 하나의 운영체제에 있어서, 그의 CPU 스케줄링의 방식은 하기와 같다. Linux 시스템의 태스크(즉, 서비스)는 일반적으로 태스크 큐에서 유지보수하고, 큐에서의 어떠한 프로세스(태스크와 대응

됨)가 CPU에 의해 실행될 수 있는지에 대해 프로세스 스케줄러에서 결정하며, 동시에, Linux 시스템에서의 할당 프로그램 모듈은 프로세스 스케줄러로부터 선택된 프로세스에 CPU의 제어 권한을 부여한다. 선택된 CPU는 Linux 시스템에서의 CPU 리소스풀(즉, 상기 사용 가능한 리소스풀)로부터 선택된 것으로서, Linux 시스템에서 리소스풀에서의 이러한 CPU를 스케줄링 하여 프로세스의 실제 실행에 참여시킬 수 있는 것과 같다.

- [0394] 프로세스 스케줄러는 실행 중인 하나의 프로세스를 일시 정지하고, 상기 프로세스를 running queue(즉, 실행 큐)에 이전하여, 다른 프로세스에 대한 실행을 시작할 수 있다. 이는 클록 중단, I/O(Input/Output, 입력/출력) 중단, 시스템에서 다른 종류의 신호를 호출한 후, 어느 하나가 준비 큐에 있을지, 또한 메인 메모리에서의 프로세스가 CPU에 의해 실행(CPU의 제어 권한을 이 프로세스에 부여함)될 수 있을지를 결정할 수 있다.
- [0395] 본 실시예에서, 스케줄링된 처리 리소스를 대응되는 운영체제의 사용 가능한 리소스풀에 병합하는 단계를 통해, 쉽게 처리 리소스에 대한 스케줄링을 수행할 수 있고, 처리 리소스 스케줄링의 효율을 향상시킬 수 있다.
- [0396] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S232을 더 포함한다.
- [0397] S232에서, 프로세서가 위치하는 칩의 전원이 켜진 후, 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하고, 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제의 실행을 부트한다. 여기서, 제1 초기 처리 리소스는 프로세서의 처리 리소스에서 제1 운영체제와 대응되는 초기 처리 리소스이고, 제2 초기 처리 리소스는 프로세서의 처리 리소스에서 제2 운영체제와 대응되는 초기 처리 리소스이다.
- [0398] 전반 시스템은 작동 시간대에 따라 2개 단계로 구분될 수 있다. 즉, 초기 시작 단계 및 실시간 실행 단계로 구분될 수 있다. 상기 처리 리소스의 동적 스케줄링 등 과정은 실시간 실행 단계에서 실행되는 것이다. 초기 시작 단계의 경우, 초기 시작 단계는 전원이 켜질 때부터 시작된다. 즉, 프로세서가 위치하는 칩의 전원이 켜질 때부터 시작된다. 시스템의 전원이 켜진 후 하나의 코어를 웨이크업하여 운영체제의 부트 동작을 실행하도록 하며, 나머지 코어는 잠시 휴면 상태에 있다. 전원이 켜진 후 시스템에서는 먼저 사전 설정된 코어 스케줄링 전략(시작 부트 전략)을 실행한다. 즉, 프로세스의 하나의 코어에서 코어 스케줄링 전략을 실행한다. 코어 스케줄링 전략은 SOC 슬라이스에서의 RAM 또는 Norflash(비휘발성 플래시 메모리)에 저장된다. 상기 스케줄링 전략은 서로 다른 디자인 수요에 따라 원활하게 구성될 수 있으며, 그의 주요한 기능은, 서로 다른 운영체제에서 실행되어야 할 초기 처리 리소스를 지정하고(예를 들면, 제1 운영체제와 제2 운영체제에서 실행되어야 할 초기 코어를 지정), 이기중 운영체제의 부트 과정을 결정하는 것을 포함한다.
- [0399] 프로세서의 처리 리소스에서, 제1 운영체제와 대응되는 초기 처리 리소스는 제1 초기 처리 리소스이고, 제2 운영체제와 대응되는 초기 처리 리소스는 제2 초기 처리 리소스이다. 시스템의 전원이 켜진 후, 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하고, 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제에 대한 실행을 부트한다. 여기서, 부트 로드 프로그램(boot loader)은 컴퓨터 또는 다른 컴퓨터 응용에 위치하는 것으로서, 운영체제를 부트 로드하는 프로그램을 가리킨다.
- [0400] 여기서, 칩은 BMC(Baseboard Management Controller, 기판 관리 제어기, 서버 플랫폼에서의 관리 제어칩) 칩일 수 있으며, 이는 ARM 멀티 코어 아키텍처에 기반하는 SOC 칩으로서, 다양한 주변 하드웨어 IP(Intelligent Peripheral, 지능 주변기기)를 통합한 것을 가리킨다. BMC 칩은 메인보드(서버 마더보드)에 설치되고, 칩의 전원이 켜지는 것은 SOC 칩 분야에서 전원이 켜지는 것을 가리킨다.
- [0401] 시스템의 전원이 켜진 후, 먼저 프로세서의 지정 프로세서 코어를 웨이크업할 수 있다. 상기 지정 프로세서 코어는 부트 로드 프로그램을 통해 각 운영체제가 대응되는 초기 처리 리소스에서 시작되도록 부트한다. 부트 로드 프로그램은 운영체제의 로딩을 부트하기 위한 프로그램이다. 예를 들면, Bootrom(CPU 슬라이스에서의 ROM)에서의 고유한 프로그램이다. 고유한 프로그램은 운영체제 시작을 부트하는 코드를 가리키며, 이는 BootLoader 프로그램에 속한다.
- [0402] 본 실시예에서, 초기 시작 단계에서, 부트 로드 프로그램을 통해 각 운영체제에 대응되는 초기 처리 리소스에서 각 운영체제를 부트 시작하는 단계를 통해, 운영체제 시작의 성공율을 향상시킬 수 있으며, 동시에 실시간 실행 단계를 위해 준비할 수 있다.
- [0403] 예시적인 일 실시예에서는, 제1 초기 처리 리소스는 프로세서의 프로세서 코어 중의 하나의 지정 프로세서 코어이고, 제2 초기 리소스는 프로세서의 프로세서 코어 중 지정 프로세서 코어 이외의 다른 프로세서 코어이다.
- [0404] 본 실시예에서는, 초기 처리 리소스는 초기 프로세서 코어를 포함할 수 있다. 제1 초기 처리 리소스는 프로세서

의 프로세서 코어 중의 적어도 하나의 프로세서 코어일 수 있고, 제2 초기 처리 리소스는 제1 초기 처리 리소스 이외의 다른 프로세서 코어의 적어도 일부분의 프로세서 코어일 수 있다. 선택적으로, 시스템의 시작 효율을 보장하기 위해, 제1 초기 처리 리소스는 프로세서의 프로세서 코어 중의 하나의 지정된 프로세서 코어일 수 있고, 제2 초기 처리 리소스는 프로세서의 프로세서 코어 중 지정 프로세서 코어 이외의 다른 프로세서 코어일 수 있다.

- [0405] 예를 들면, 8코어 프로세서(CPU0 내지CPU7)의 경우, 지정 프로세서 코어는 CPU0(또는 다른 어느 한 프로세서 코어)일 수 있다. 즉, 제1 운영체제의 초기 처리 리소스는 CPU0이고, 제2 운영체제의 초기 처리 리소스는 CPU1 내지 CPU7이다.
- [0406] 본 실시예에서, 어느 한 운영체제의 초기 처리 리소스를 지정하여 하나의 지정된 프로세서 코어로 하고, 프로세서의 다른 프로세서 코어를 다른 운영체제의 초기 처리 리소스로 하는 단계를 통해, 시작 부트의 편의성을 향상시킬 수 있다.
- [0407] 예시적인 일 실시예에서는, 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제를 부트 실행하는 단계는 하기와 같은 단계 S241를 포함한다.
- [0408] S241에서, 2차 프로그램 로더를 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트한다.
- [0409] 본 실시예에서는, 2차 프로그램 로더(Second Program Loader, SPL로 약칭함)를 통해 운영체제의 시작에 대한 부트를 수행한다. SPL은 uboot의 제1 단계에서 실행되는 코드에 속하고, uboot의 제2 단계를 메모리에 이동하여 실행할 수 있으며, 주로 운영체제를 RAM에서의 프로그램 코드에 로딩하는 것을 담당한다. 여기서, uboot는 임베디드 시스템의 부트 로드용 위한 프로그램으로서, 주로 임베디드 시스템의 부트 로드 프로그램에 사용되고, 예를 들면, PPC, ARM, AVR32, MIPS, x86, 68k, Nios 및 MicroBlaz 등과 같은 다양한 서로 다른 컴퓨터 시스템 구조를 지원할 수 있다.
- [0410] 본 실시예에서, 2차 프로그램 로더를 통해 초기 처리 리소스에서 대응되는 운영체제의 실행을 부트하는 단계를 통해, 기존의 시스템 시작 흐름에 호환되어, 시스템 시작의 효율을 향상시킬 수 있다.
- [0411] 예시적인 일 실시예에서는, 상기 방법은 하기와 같은 단계 S251를 더 포함한다.
- [0412] S251에서, 기기의 메인보드에 고정된 시작 칩에서의 사전 설정된 프로그램을 통해 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행한다. 여기서, 2차 프로그램 로더는 보안 시작 점검의 점검 결과가 정상인 후에 실행된다.
- [0413] 2차 프로그램 로더를 사용하여 운영체제의 실행을 부트하기 전에, 운영체제 시작의 성공율을 향상시키기 위해, 기기 메인보드에 고정된 시작 칩에서의 사전 설정된 프로그램은 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행하여, 2차 프로그램 로더가 정상인지 여부를 결정할 수 있다. 2차 프로그램 로더에 대한 보안 시작 점검의 결과가 정상일 경우, 2차 프로그램 로더를 통해 운영체제의 실행을 부트할 수 있고, 2차 프로그램 로더에 대한 보안 시작 점검의 결과가 예외일 경우, 예외 알림을 수행하고, 운영체제의 부트 과정을 종료한다.
- [0414] 예를 들면, 시스템의 전원이 켜진 후, 일부분의 코어를 구성하여 Bootrom에서의 고유한 프로그램을 부트하고, Bootrom 프로그램(한세트의 기기 메인보드에 고정된 ROM 칩에서의 프로그램)을 통해 SPL 일부분의 코드에 대해 보안 시작 점검을 수행하며, 점검 결과가 정상일 경우 SPL 코드를 실행한다.
- [0415] 본 실시예에서, 2차 프로그램 로더를 통해 보안 시작 점검을 수행하는 단계를 통해, 운영체제 시작의 보안성을 향상시킬 수 있다.
- [0416] 예시적인 일 실시예에서는, 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제가 실행을 부트하는 단계는 하기와 같은 단계 S261을 포함한다.
- [0417] S261에서, 2차 프로그램 로더를 통해 제2 초기 처리 리소스를 웨이크업하고, 범용의 부트 로드 프로그램의 실행을 부트하여, 제2 운영체제의 실행을 부트한다.
- [0418] 제2 운영체제의 실행을 부트할 경우, 2차 프로그램 로더를 통해 먼저 제2 초기 처리 리소스를 웨이크업한 다음, 범용의 부트 로드 프로그램의 실행을 부트하여, 제2 운영체제의 실행을 부트한다. 예를 들면, SPL은 일 측면에서는 일부분 코어(예를 들면, 코어1, CPU0)에서 제1 운영체제의 실행을 부트하고, 다른 일 측면에서는 다른 일부분의 코어(예를 들면, 코어2 내지 코어M+N, CPU1 내지 CPU M+N-1)를 웨이크업하여 uboot 코드(주로 임베디드 시스템의 부트 로드 프로그램을 위한 것임)의 실행을 부트함으로써 제2 운영체제의 실행을 부트한다. 상기 단계

를 통해, 서버 관리 제어 유닛 BMC 시스템은 제1 운영체제와 제2 운영체제의 각자가 하나 또는 복수의 프로세서 코어에 기반하여 정상적인 부트를 실현하여, 실시간 실행 단계를 위해 준비할 수 있다.

[0419] 본 실시예에서, 먼저 초기 처리 리소스를 웨이크업하고, 범용의 부트 로드 프로그램의 실행을 부트하여, 대응되는 운영체제의 실행을 부트하는 단계를 통해, 시스템의 실행을 부트하는 효율을 향상시킬 수 있다.

[0420] 아래는 선택적인 예시와 함께 본 실시예에서의 임베디드 시스템의 실행 방법을 해석하여 설명한다. 본 선택적인 실시예에서는, 제1 운영체제는 RTOS 시스템이고, 제2 운영체제는 Linux 시스템(다른 비실시간 운영체제일 수도 있음)이며, 처리 리소스는 프로세서 코어이다.

[0421] 임베디드 시스템에서 다양한 유형의 서비스를 정상적으로 실행하고, 동시에 프로세서 코어의 이용율을 향상시키며, 서버의 하드웨어 비용을 합리적인 범위내로 제어하기 위해, 본 선택적인 실시예에서는 임베디드 이기종 멀티 시스템 간의 프로세서 리소스에 대한 동적 밸런스 스케줄링 방법을 제공한다. 멀티 코어 프로세서의 하드웨어 코어에 기반하여, 코어 적응형 스케줄링(즉, 리소스 적응형 스케줄링) 방법에 따라 서로 다른 서비스 유형을 서로 다른 코어 세트에 스케줄링하여 실행한다. 동시에, 서비스 관리, 로드 밸런스, 리소스 동적 할당 및 코어 간 통신 인터페이스 등 시스템 아키텍처 디자인을 유입하여, 실시간과 비실시간 임베디드 이기종 운영체제가 동일한 프로세서 복수의 코어에서 병행하여 실행되는 효과를 실현하여, 프로세서 코어의 전반 이용율을 향상시킬 수 있고, 실시간성 서비스에 대응하기 위해 도입되는 하드웨어 가속 리소스 측면의 투입을 크게 감소할 수 있으며, 동시에, 프로세서 멀티 코어에서 고속의 내부 버스를 적용하여, 통신 성능을 크게 향상시킬 수 있고, 디자인 비용을 현저하게 감소할 수 있으며, 계산 성능을 향상시킬 수 있다.

[0422] 서버 플랫폼(멀티 코어 SOC에 사용되는 임베디드 제품에는 모두 적용됨, 예를 들면, 핸드폰, 태블릿PC, 셋톱박스 등에 마찬가지로 적용됨)에서의 BMC 관리 유닛을 예로 들면, 본 선택적인 실시예에서의 임베디드 시스템의 실행 방법은 2개 단계, 즉 초기 시작 단계 및 실시간 실행 단계로 나눌 수 있다. 도 7에 도시된 임베디드 이기종 멀티 시스템 간의 프로세서 리소스에 대한 동적 밸런스 스케줄링의 실현 아키텍처를 참조한다. 여기서, 초기 시작 단계에서 CPU 밸런스 스케줄링 과정은 도 8에 도시된 바와 같다. 상기 과정은 하기와 같은 단계 S802, S804, S806, S808 및 S810을 포함할 수 있다.

[0423] 단계 S802에서, 일부분의 코어를 구성하여 Bootrom에서의 고유한 프로그램을 부트한다.

[0424] 단계 S804에서, Bootrom 프로그램(즉, 상기 고유한 프로그램)을 통해 SPL(NorflashDP에 저장됨)에 대해 보안 시작 점검(보안 시작 점검은 주로 부트될 uboot, Linux, RTOS 이미지에 대해 해시체크를 수행하여, 획득한 결과와 저장된 결과를 비교하여, 일치할 경우, 정상적으로 부트하고, 일치하지 않을 경우, 시작을 실패하며, SPL은 일부분의 CPU 레지스터에 대해 구성하여, 어떠한 코어를 웨이크업하고, 어느 주소로부터 시스템 이미지를 로딩할지를 선택함)을 수행한다.

[0425] 단계 S806에서, 보안 점검이 통과되었는지 여부를 판단하며, 보안 점검이 통과될 경우, 단계 S808을 실행하고, 보안 점검이 통과되지 않을 경우, 단계 S810을 실행한다.

[0426] 단계 S808에서, SPL 코드를 실행하며, SPL은 일 측면에서는 일부분 코어에서 RTOS 시스템의 실행을 부트하고, 다른 일 측면에서는 다른 일부분의 코어를 웨이크업하여 uboot 코드의 실행을 부트하여, Linux 시스템의 실행을 부트한다.

[0427] 단계 S810에서, 종료된다.

[0428] 이기종 운영체제가 부트된 후, 시스템은 실시간 실행 단계에 진입한다. 도 7을 참조하면, 실시간 실행 단계는 로드 밸런스 전략, 서비스 관리, 리소스 동적 할당, 코어 적응형 스케줄링(즉, 리소스 적응형 스케줄링), 코어 선점과 해제, 및 코어 간 통신 등 기능 모듈이 협작하여 완성된다. 도 7 및 도 9를 결합하여, 실시간 실행 단계에서는, 본 선택적인 실시예에서의 임베디드 시스템의 실행 방법의 흐름은 하기와 같은 단계 단계 S902, S904, S906 및 S908를 포함한다.

[0429] 단계 S902에서, 서비스 관리 모듈은 로드 밸런스 전략 모듈의 출력에 따라, 할당 대상인 서비스의 실제 분류를 실행하여, 실시간 서비스를 포함하는 리스트와 비실시간 서비스를 포함하는 리스트를 생성한다.

[0430] 로드 밸런스 전략 모듈은 서비스 관리 모듈에게 전략 지침을 제공함으로써, 시스템에서 실행되는 다양한 서비스(또는 프로세스) 분류 방법, 실시간성 등급 평가 원칙 등을 포함한다. RTOS 시스템 및 Linux 시스템의 시작을 부트한 후, 서비스 관리 모듈은 로드 밸런스 전략 모듈의 출력에 따라, 할당 대상인 서비스의 실제 분류를 실행하여, 실시간 서비스를 포함하는 리스트와 비실시간 서비스를 포함하는 리스트를 생성한다. 로드 밸런스 전략

모듈에 의해 출력되는 2개의 리스트는 서비스와 운영체제 사이의 대응관계를 나타내기 위한 것이다.

- [0431] BMC 응용의 경우, 서버 플랫폼에 대한 강력하고 전반적인 관리를 실행하기 위해, 서버 플랫폼에서는 대량의 서비스 프로세스가 필요하고, 이때, 서비스 관리 모듈은 이러한 서비스에 대해 합리적인 분류를 실현한다. 한가지 가능한 분류 방식은, 서버의 건강 상태를 모니터링, IPMI(Intelligent Platform Management Interface, 지능 플랫폼 관리 인터페이스) 인터랙션, 부품 모니터링, 직렬포트 리디렉션, 자산 정보 관리, Web 액세스 등 서비스를 비실시간 서비스로 구분하는 방식; 팬 제어, PECI(Platform Environment Control Interface, 플랫폼 환경식 제어 인터페이스) 통신 등의 서비스를 실시간 서비스로 구분하는 방식; 이다.
- [0432] 단계 S904에서, 리소스 동적 할당 모듈은 서비스 관리 모듈의 출력 결과에 따라, 프로세서 리소스(프로세서 하드 코어와 제어기 로직 유닛 등)를 동적 조정하여, 서로 다른 서비스와 실제 하드웨어 리소스의 리소스 매핑 테이블을 형성한다.
- [0433] 리소스 동적 할당 모듈에 의해 형성되는 매핑 테이블은 서비스와 프로세서 리소스의 리소스 매핑 테이블이다. 즉, 할당된 각 운영체제의 프로세서 리소스를 포함하고, 어느 한 운영체제에도 할당되지 않은 프로세서 리소스도 포함한다. 서비스와 실제 하드웨어 리소스 사이의 매핑을 수행할 시, 실시간 서비스 할당과 RTOS 시스템의 프로세서 코어를 우선으로 할 경우, RTOS 시스템의 프로세서 코어에 모두 하나 또는 복수의 서비스가 할당된 후, 실시간 서비스에 할당되지 않은 프로세서 코어를 할당하는 것을 고려하고, 비실시간 서비스 할당과 Linux 시스템의 프로세서 코어를 우선으로 할 경우, Linux 시스템의 프로세서 코어에 모두 하나 또는 복수의 서비스가 할당된 후, 비실시간 서비스에 할당되지 않은 프로세서 코어를 할당하는 것을 고려한다. 따라서, 실시간 서비스 또는 비실시간 서비스에 할당되지 않은 프로세서 코어가 존재한다.
- [0434] 실제 응용에서는, BMC의 프로세서 코어 및 주변기기 제어기 등 하드웨어 리소스를 BMC 실제 서비스와 동적으로 매핑한다. 서버를 견후 BMC에서의 코어1로 RTOS 시스템을 실행하고, PWM(Pulse Width Modulation, 펄스 폭 변조) 및 PECI 주변기기 제어기를 제어하여 팬 속도 조절 및 고장 진단 서비스를 실현하며, 코어2 내지 코어N으로 Linux 시스템을 실행하여 I2C(Inter-Integrated Circuit, 집적 회로 버스) 등 저제어를 제어하여 부품 모니터링 서비스를 실현한다. 시스템 실행 과정에서, 시스템 로드 등 상황에 따라 BMC 리소스의 동적 할당 테이블을 동적으로 생성하여, 다음 코어 적응형 스케줄링 과정을 위해 준비한다.
- [0435] 여기서, 출력되는 리소스 매핑 테이블은 실시간 서비스의 X개 코어 및 xxx 제어기(어떠한 유형의 제어기), 비실시간 서비스의 Y개의 코어 및 xxx제어기(어떠한 유형의 제어기)일 수 있다. 어떠한 서비스는 컴퓨터 리소스(대응되는 프로세서 코어)에 의존할 뿐만 아니라, 특정의 하드웨어 인터페이스(대응되는 SOC에서의 제어기)에도 의존한다. 예를 들면, 이더넷 포트가 사용되어야 할 서비스에는 상기 서비스와 대응되는 네트워크 제어기를 동일한 운영체제(Linux 시스템 또는 RTOS 시스템)로 분할해야 된다.
- [0436] 단계 S906에서, 코어 적응형 스케줄링 모듈은 하드웨어 리소스의 동적 할당 결과에 따라, 프로세서 하드 코어 리소스에 대한 실제 스케줄링 동작을 완성한다.
- [0437] 코어 적응형 스케줄링 모듈은 서비스 관리 모듈의 출력 결과 및 리소스 동적 할당 모듈의 출력 결과에 따라 프로세서 하드 코어 리소스에 대한 실제 스케줄링을 수행한다. 예를 들면, 도 6에 도시된 코어세트1의 M개의 코어와 같이 일부분의 프로세서 하드 코어를 스케줄링하여 실시간성 서비스를 실행하고, 도 6에 도시된 코어세트2의 N개의 코어와 같이 나머지 프로세서 하드 코어를 스케줄링하여 비실시간성 서비스를 실행한다. 전반 스케줄링 과정은 Linux 시스템에서 주도하고, 구체적으로는 Linux의 SMP 프레임과 협력하여 프로세서 하드 코어 리소스풀에 대해 구성하여 실현할 수 있다. 여기서, 프로세서의 사용 가능한 총 코어 수는 $M+N$ 이며, $M \geq 1$ 이고, $N \geq 1$ 이다.
- [0438] 단계 S908에서, 시스템 실시간 실행 과정에서, 코어 선점과 해제 모듈은 이기종 운영체제 사이에서 동적으로 프로세서 코어에 대한 선점과 해제를 수행한다.
- [0439] 선점과 해제 과정은 이기종 운영체제의 실시간 로드 상황에 따라 결정되며, 프로세서 리소스 이용율의 최대화 목표에 달성한다. 예를 들면, BMC에 원격으로 액세스하는 사용자 수가 1에서 20으로 증가할 경우, BMC의 Web 액세스 서비스량은 급증하여, Linux 시스템에 더 많은 프로세서 하드 코어의 지원이 필요하게 된다. 이러한 경우, Linux 시스템은 IPI를 통해 RTOS 시스템으로 코어 선점 요청을 발신한다. 이때 RTOS 시스템이 유휴 상태(즉, 스케줄링할 태스크가 없음)일 경우, 이는 Linux 시스템에 코어 리소스를 해제하고, 이때 Linux 시스템은 코어 선점에 성공한다. 즉, Linux는 더 많은 프로세서 리소스를 획득하여, 새로 증가된 서비스 수요에 대응한다. 코어 선점과 해제 메커니즘을 통해, 서버 원격 관리 측면의 사용자 체험이 현저하게 향상되도록 한다.

- [0440] 코어 선점과 해제 과정은 하나의 동적으로 순환 왕복되는 과정이다. 매회 코어 선점과 해제는 프로세서 하드 코어 리소스의 조정이 수반한다. 도 6에 도시된 바와 같이, 1회의 선점과 해제를 거쳐, 코어세트X의 코어 수는 P로 되고, 코어세트Y의 코어 수는 M+N-P로 된다. 또한, 코어 선점과 해제 과정은 로드 밸런스 전략, 서비스 관리, 리소스 적응형 스케줄링 등 모듈과 협동하여 최종의 코어 조정 결과를 함께 결정할 수 있다.
- [0441] 상기 순환 왕복은, 시스템이 정상으로 실행되는 과정에서, RTOS 시스템과 Linux 시스템에서 일측 리소스의 부족으로 인해 리소스 선점과 해제 동작을 수행하는 상황이 끊임없이 발생하는 것을 가리킨다. 상기 순환이 실행되는 과정은 “서비스 관리(서비스 관리는 동적으로 구분되는 과정)-하드웨어 리소스 동적 할당-코어 적응형 스케줄링” 등 모듈의 협동 실행에 관련된다. 상기 복수의 모듈이 모두 비실시간 시스템(Linux 시스템)에서 실행되는 것은, 대부분의 주요한 서비스가 모두 Linux 시스템에서 실행되어, 더 많은 정보 및 상태를 더 쉽게 획득하고 감지할 수 있도록 한다.
- [0442] 상기 복수의 모듈에서 코어 조정 결과를 함께 결정하는 것은, 어느 한 시스템에서 더 많은 코어 리소스를 선점해야 할 경우, 최적적인 선점 성공 여부는 로드 밸런스, 서비스 관리, 코어 적응형 조정 등 모듈에서 함께 결정해야 하는 것을 가리킨다. 예를 들면, 그 중 한가지 가능한 상황에서, Linux 시스템에서 더 많은 리소스가 필요할 경우, 이때 RTOS 시스템가 유휴 상태(휴면 상태)이지만, RTOS 시스템에서 곧 하나의 중요한 서비스 스레드를 웨이크업하여 실행해야 하고, 또한 상기 스레드 실행 시 많은 리소스가 필요하므로, 이때 RTOS 시스템에서는 Linux 시스템에 코어 리소스를 해제할 수 없다. 상기 리소스 스케줄링 과정은 서비스 중요성을 판단하는 기준 및 최종적으로 리소스를 Linux 시스템에 할당하지 않는 동작 등은 모두 로드 밸런스 전략, 서비스 관리, 코어 적응형 조정 등 모듈의 참여에 관련된다. 이러한 모듈 각자의 전략은 개방성 특성을 포함하고, 이는 이들에 의해 함께 결정된 최종 핵심 조정 결과도 이러한 전략이 하계 작용한 결과임을 결정한다.
- [0443] 여기서, RTOS 시스템이 휴면한 후 CPU 코어 리소스를 전혀 점유하지 않으므로, 상기 RTOS 시스템이 웨이크업될 경우(예를 들면, 주변기기 또는 타이머를 웨이크업하여 중단이 웨이크업을 트리거하는 동작을 생성함), 코어 간 중단을 통해 지정된 코어 리소스(예를 들면, 코어0)의 제어 권한을 다시 획득할 수 있다. 상기 휴면 상태에서부터 코어 제어 권한을 다시 획득하기까지 RTOS 시스템은 모두 코어의 참여가 필요하지 않다.
- [0444] 아래는 8코어 프로세서(코어1 내지 코어8)를 예로 들어 임베디드 이기종 멀티 시스템 간의 프로세서 리소스에 대한 동적 밸런스 스케줄링 방법을 해석하여 설명한다. 실시간 운영체제와 비실시간 운영체제가 각각 RTOS 시스템과 Linux 시스템인 것으로 가정할 경우, 프로세서 리소스에 대한 동적 밸런스 스케줄링 흐름은 하기와 같은 단계 1, 단계 2, 단계 3, 단계 4 및 단계 5를 포함할 수 있다.
- [0445] 단계 1에서, 시스템의 전원을 켜고, 사전 설정된 시작 부트 전략을 실행하며, 코어1(CPU0)에 기반하여 RTOS 시스템을 부트하고, 코어8(CPU7)에 기반하여 Linux 시스템을 부트한다.
- [0446] 단계 2에서, Linux 시스템의 서비스 관리 모듈은 로드 밸런스 전략 모듈에 기반하여 시스템 서비스에 대한 실시간성 등급 구분을 실현하여, 실시간성 서비스 리스트와 비실시간성 서비스 리스트를 획득한다. 서비스가 총 20개 있다고 가정할 경우 실시간성 서비스는 서비스1과 서비스2이고, 비실시간성 서비스는 서비스3 내지 서비스20이다.
- [0447] 단계 3에서, 서비스 관리 모듈에 의해 출력되는 서비스 리스트에 따라 리소스 동적 할당 모듈과 코어 적응형 스케줄링 모듈을 시작하고, Linux 시스템의 SMP 프레임워크를 통해 8개의 프로세서 코어를 프로세서 코어세트1(코어1과 코어2를 포함한다고 가정함)과 코어세트2(코어3 내지 코어8을 포함한다고 가정함)로 구분하며, 이때, 시스템에서는 코어세트1의 리소스를 RTOS 시스템에 할당하여 서비스1과 서비스2를 실행하고, 동시에 코어세트2의 리소스를 Linux 시스템에 할당하여 서비스3 내지 서비스20을 실행한다.
- [0448] 단계 4에서, 시스템을 실행하는 과정에서, RTOS 시스템과 Linux 시스템은 모두 각자의 로드 상황을 모니터링하고, 어느 일 측의 서비스 로드가 가중할 경우, 코어 간 통신 인터페이스를 통해 코어 선점 동작을 발신하며, 다른 일 측에서 코어 선점 요청을 수신할 경우, 자체의 로드 상황에 따라 상대측에 하드 코어 리소스를 해제할지 여부를 판단한다.
- [0449] 예를 들면, Linux 시스템에서는 어느 한 시각에 자체의 로드가 급증하는 것이 모니터링될 경우, 자체의 계산 리소스가 완전히 소모되기 전에 코어 간 통신 인터페이스를 통해 RTOS 시스템으로 코어 선점 중단 요청(표 1에서 중단 번호가 9인 SGI 중단에 대응됨)을 발신한다. RTOS 시스템에서는 코어 선점 중단 요청을 수신한 후 자체에 의해 점유된 코어의 유휴 상태 여부를 점검하며, 이때 유휴한 코어(예를 들면, 코어 2가 유휴함)가 존재할 경우, 코어2를 해제하고, 코어 간 통신 인터페이스를 통해 Linux 시스템에 코어 해제 중단 응답 신호(표 1에서

중단 번호가 10인 SGI 중단에 대응됨)를 발신하며, Linux 시스템에서는 중단 응답 신호를 수신한 후 SMP 프레임 을 통해 코어2를 사용 가능한 리소스풀에 병합하고, 이때 프로세서 하드 코어는 1회의 새로운 조정을 실현한다. 즉, RTOS 시스템에서 코어1을 점유하고, 나머지 7개의 코어는 Linux 시스템에서 점유한다.

[0450] 단계 5에서, 시스템 로드 상황을 업데이트하고, 단계 2로 반환한다.

[0451] 관련 기술에서의 멀티 코어 싱글 시스템 및 멀티 코어 고정형 멀티 시스템(각 코어가 어느 한 특정 운영체제를 고정 실행함)과 달리, 본 선택적인 실시예에 의해 제공되는 방법은 실제 서비스 수요에 따라, 서로 다른 하드웨어 프로세서 코어를 실시간 운영체제 및 비실시간 운영체제에 동적 할당하여, 소프트웨어 및 하드웨어 시스템 아키텍처 분야에서 프로세서 하드 코어에 기반하는 임베디드 이기종 멀티 시스템 간의 프로세서 리소스에 대한 동적 스케줄링을 디자인하는 것을 실현한다. 프로세서 하드 코어는 동적 할당성을 가져, 시스템에 병존하는 이 기종 운영체제에서 자체의 서비스 로드 상황에 따라 합리적인 프로세서 계산 리소스를 획득할 수 있도록 함으로써, 일 측면에서 프로세서에서 실시간 서비스에 대한 처리 능력을 향상시키고, 동시에 프로세서 코어의 이용율을 전면적으로 향상시키며, 계산 성능을 향상시켜, 하드웨어 비용 절약의 목적에 도달한다.

[0452] 선택적인 일 응용 장면에서는, 아래에서 2개의 운영체제가 교체하여 동일한 기기(즉, 타깃 기기)의 실행 상태를 제어하는 응용 장면과 함께 본 실시예에서의 임베디드 시스템의 실행 방법을 해석하여 설명한다. 여기서, 제1 운영체제의 처리 리소스는 프로세서의 제1 프로세서 코어를 포함하고, 제2 운영체제의 처리 리소스는 프로세서의 제2 프로세서 코어를 포함한다. 본 실시예에서는, 운영체제의 시작 제어 과정을 제공한다. 상기 과정은 하기와 같은 단계 A, B 및 C를 포함한다.

[0453] 단계 A에서, 프로세서의 제1 프로세서 코어에서 실행되는 제1 운영체제를 통해, 제2 버스를 거쳐 타깃 기기의 하드웨어 제어기를 제어하여, 타깃 기기의 실행 상태를 제어한다.

[0454] 서버, 개인용 컴퓨터, 산업용 퍼스널 컴퓨터 등 기기의 경우, 일부 특정 기기를 배치하여 기기 실행과 관련되는 동작을 실행할 수 있다. 관련 기술에서는, 통상적으로 시스템의 전원이 켜진 후, 이러한 특정 기기가 작동되기 시작한다. 시스템의 전원이 켜진 후, 프로세서에서 실행되는 운영체제는 일정한 시간 동안을 거쳐야만 특정 기기를 정상적으로 연결 관리하여, 특정 기기의 실행 상태를 제어하며, 운영체제가 시작되는 과정에서는 특정 기기에 대한 제어가 안된다.

[0455] 예를 들면, 시스템의 전원이 켜진후 팬이 작동되기 시작하고, 시스템의 전원이 켜진 후 CPU에서 실행되는 운영체제는 일정한 시간 동안을 거쳐야만 팬을 정상적으로 연결 관리하여, 팬의 회전속도를 설정하므로, 운영체제가 시작되는 과정에서는 팬에 대한 제어가 어렵다.

[0456] 예를 들면, 운영체제가 시작되는 과정에서 팬에 대한 제어를 실현하기 위해, 서버에서는 BMC와 CPLD를 결합하는 제어 방식을 적용한다. 개인용 컴퓨터는 EC 칩의 제어 방식(EC 칩에서 온도에 따라 팬의 회전속도를 조정하는 기능임)을 적용하고, 산업용 퍼스널 컴퓨터는 맞춤형 칩의 제어 방식을 적용한다. 서버, 개인용 컴퓨터, 산업용 퍼스널 컴퓨터의 운영체제가 시작되는 과정에서, CPLD, EC 칩, 맞춤형 칩은 팬의 회전속도를 제어하는데 개입하게 된다. 운영체제가 시작 완료 후, 팬에 대한 제어 권한은 운영체제에서의 응용 프로그램에 부여된다.

[0457] 적어도 일부분의 상기 기술적 과제를 해결하기 위해, 멀티 코어 멀티 시스템(예를 들면, 멀티 코어 듀얼 시스템)의 시작 제어 방식을 적용할 수 있다. 프로세서의 서로 다른 프로세서 코어에서 임베디드 시스템의 서로 다른 운영체제를 실행할 경우, 서로 다른 운영체제의 응답 속도는 서로 다르다. 제2 운영체제가 시작되지 않거나, 재부팅되거나 또는 특정 기기의 실행 상태를 제어할 수 없는 다른 상황의 경우, 응답 속도가 제2 운영체제보다 빠른 제1 운영체제에서 특정 기기의 실행 상태를 제어할 수 있어, 특정 기기의 실행 상태에 대한 제어가 어려운 상황을 줄일 수 있고, 동시에, 별도의 비용을 증가할 필요가 없으며, 또한 우수한 확장 가능성도 갖는다.

[0458] 본 실시예에서는, 제2 운영체제가 시작되지 않거나, 재부팅되거나 또는 타깃 기기의 실행 상태를 제어할 수 없는 다른 상황의 경우, 제1 운영체제를 통해 제2 버스를 거쳐 타깃 기기의 하드웨어 제어기를 제어하여, 타깃 기기의 실행 상태를 제어할 수 있다. 여기에서의 타깃 기기는 팬이거나, 또는 시스템 시작 시 실행되어야 하는 다른 기기일 수 있다. 팬의 경우, 이에 대응되는 하드웨어 제어기는, 예를 들면, PWM(Pulse Width Modulation, 펄스 폭 변조) 제어기, FanTach(팬 회전속도) 제어기와 같은 팬 제어기이다. 여기서, 제1 운영체제(예를 들면, RTOS 시스템)를 사용하여 종래의 CPLD, EC 칩, 맞춤형 칩을 대체하여, 일 측면에서는 하드웨어 비용을 절약하고, 다른 일 측면에서는 기기 제어는 소프트웨어로 실현되는 것이므로, 확장 가능성이 높다.

[0459] 예를 들면, BMC 듀얼 코어에 기반하여 RTOS 시스템 및 Linux 시스템의 듀얼 시스템을 실현하고, 멀티 코어 듀얼

시스템에 기반하여 팬을 실현하며, RTOS 시스템의 실시간성이 높은 특성을 이용하여, Linux 시스템이 시작되는 과정에서, RTOS 시스템으로 CPLD, EC 칩, 맞춤형 칩을 대체하여 팬을 제어할 수 있다. 즉, 팬의 제어 권한을 부여받아, 충분히 빠른 속도로 팬의 실행 상태를 제어할 수 있다.

[0460] 단계 B에서, 프로세서의 제2 운영체제 코어에서 제2 운영체제가 시작되도록 부트한다.

[0461] 시스템의 전원이 켜질 시 또는 제2 운영체제가 재부팅될 시, 프로세서의 제2 프로세서 코어에서 제2 운영체제의 시작을 부트하여, 제2 운영체제가 제2 프로세서 코어에서 실행되도록 할 수 있다. 여기서, 제2 프로세서 코어에서 제2 운영체제가 시작되는 것은 제2 프로세서 코어를 제2 운영체제에 스케줄링하는 것을 가리킨다. 운영체제의 시스템 파일 또는 이미지 파일은 프로세서가 위치하는 칩 또는, 예를 들면, 외부 RAM과 같은 칩 이외의 메모리에 저장될 수 있다.

[0462] 단계 C에서, 제2 운영체제가 시작된 후, 제2 운영체제를 통해 제2 버스를 거쳐 하드웨어 제어기를 연결 관리하여, 타깃 기기의 제어 권한을 부여받는다.

[0463] 제2 운영체제가 시작 완료된 후, 줄곧 제1 운영체제에서 타깃 기기에 대한 실행 상태를 제어할 수 있으며, 멀티 코어 프로세서에서 복수의 운영체제를 실행할 경우 복수의 운영체제 사이에서 데이터 인터랙션을 수행해야 하는 것을 고려하여, 제1 운영체제에서 기기의 전반 제어를 수행할 수도 있고, 제2 운영체제에서 타깃 기기의 제어 권한을 연결 관리할 수도 있다. 예를 들면, 제2 운영체제를 통해 제2 버스를 거쳐 하드웨어 제어기를 연결 관리할 수 있다. 제2 운영체제에서 타깃 기기의 제어 권한을 연결 관리하는 방식은 하기와 같다. 제2 운영체제가 시작된 후, 제2 운영체제에서 제1 운영체제로 기기 연결 관리 요청을 발신, 예를 들면, 제1 버스를 통해 중단 요청을 발신하여, 타깃 기기의 하드웨어 제어기에 대한 연결 관리를 요청한다. 제1 운영체제는 제2 운영체제에 의해 발신된 기기 연결 관리 요청을 수신하여, 타깃 기기의 제어 권한을 제2 운영체제에 부여할 수 있고, 타깃 기기의 제어 권한을 부여하는 것과 관련되는 동작도 실행할 수 있다. 예를 들면, 타깃 기기의 실행 상태에 대한 제어를 위한 서비스(프로세스)의 실행을 정지한다.

[0464] 예를 들면, Linux 시스템이 시작 완료 후, RTOS 시스템은 팬의 제어 권한을 Linux 시스템에 부여하고, Linux 시스템에서 팬을 제어하도록 한다. 상기 과정은 시스템의 전원이 켜진 후 실행될 수 있다. 즉, 멀티 코어 듀얼 시스템의 시작 방식을 적용한다. 먼저 RTOS 시스템이 시작되면, 더 일찍 팬 제어에 개입하는데 유리하고, Linux 시스템이 완전히 시작된 후까지 대기하면, RTOS 시스템은 팬의 제어 권한을 Linux 시스템에 부여하여 Linux 시스템에서 제어하도록 한다.

[0465] 예시적인 일 실시예에서는, 프로세서의 제1 프로세서 코어에서 실행되는 제1 운영체제를 통해 제2 버스를 거쳐 타깃 기기의 하드웨어 제어기를 제어하는 단계 이전에, 프로세서가 위치하는 칩의 전원이 켜진 후, 프로세서를 통해 제1 프로세서 코어를 웨이크업하는 단계; 제1 프로세서 코어를 통해 제1 운영체제의 부트 로드 프로그램을 실행하여, 제1 운영체제가 제1 프로세서 코어에서 시작되도록 부트하는 단계; 를 더 포함한다.

[0466] 전반 시스템은 작동 시간대에 따라 초기 시작 단계 및 실시간 실행 단계의 2개 단계로 구분될 수 있으며, 본 실시예에서의 시작 제어 방법은 초기 시작 단계 또는 실시간 실행 단계에서 실행되는 것일 수 있다. 초기 시작 단계의 경우, 초기 시작 단계는 시스템의 전원이 켜질 때, 즉 프로세서가 위치하는 칩의 전원이 켜질부터 시작된다. 시스템의 전원이 켜진 후 하나의 코어는 웨이크업되어 운영체제의 부트 동작을 실행하고, 나머지 코어는 잠시 휴면 상태에 있게 된다. 웨이크업된 코어는 제1 프로세서 코어일 수 있다.

[0467] 선택적으로, 전원이 켜진 후 먼저 사전 설정된 코어 스케줄링 전략(시작 부트 전략)을 실행한다. 즉, 프로세서의 하나의 프로세서 코어에서 코어 스케줄링 전략을 실행한다. 코어 스케줄링 전략은 SOC 슬라이스에서의 RAM 또는 Norflash(비휘발성 플래시 메모리)에 저장될 수 있다. 상기 스케줄링 전략은 서로 다른 디자인 수요에 따라 원활하게 구성될 수 있으며, 그의 주요한 기능은, 서로 다른 운영체제에서 실행되어야 할 초기 처리 리소스(프로세서 코어)를 지정하고, 이기중 운영체제의 부트 과정을 결정하는 것을 포함한다. 칩의 전원이 켜지는 것은 SOC 칩 분야에서 전원이 켜지는 것을 가리킨다.

[0468] 제1 프로세서 코어가 웨이크업된 후, 부트 로드 프로그램을 통해 제1 프로세서 코어에서 제1 운영체제의 실행을 부트할 수 있고, 제1 프로세서 코어에서 부트 로드 프로그램을 통해 제1 운영체제가 제1 프로세서 코어에서의 시작을 부트할 수 있다. 부트 로드(BootLoader) 프로그램은 컴퓨터 또는 다른 컴퓨터 응용에 위치할 수 있으며, 이는, 예를 들면, BootRom에서의 고유한 프로그램과 같은 운영체제 로딩을 부트하기 위한 프로그램을 가리킨다. 고유한 프로그램은 운영체제의 시작을 부트하는 코드로서, BootLoader 프로그램에 속하고, BootRom은 CPU 슬라이스에서의 임베디드 프로세서 칩 내의 하나의 작은 마스크 ROM(Read-Only Memory, 리드 온리 메모리) 또는 리

드 온리 플래시 메모리이다.

- [0469] 초기 시작 단계에서, 부트 로드 프로그램을 통해 운영체제에 대응되는 프로세서 코드에서의 시작을 부트하여, 운영체제 시작의 성공율을 향상시킬 수 있고, 동시에 실시간 실행 단계를 위해 준비할 수 있다.
- [0470] 예시적인 일 실시예에서는, 프로세서의 제1 프로세서 코어에서 실행되는 제1 운영체제를 통해 제2 버스를 거쳐 타깃 기기의 하드웨어 제어기를 제어하는 단계는, 제1 프로세서 코어에서 제1 운영체제의 제1 제어 태스크를 실행하되, 제1 제어 태스크는 하드웨어 제어기를 제어하기 위한 것인 단계; 제1 프로세서 코어를 통해 타깃 기기와 대응되는 지정 센서의 센서 데이터를 읽는 단계; 제1 제어 태스크를 통해 지정 센서의 센서 데이터에 따라 제2 버스를 거쳐 하드웨어 제어기로 기기 제어 명령어를 발신하여, 하드웨어 제어기에서 기기 제어 명령어에 따라 타깃 기기의 실행 상태를 제어하는 단계; 를 포함한다.
- [0471] 운영체제에서 타깃 기기의 하드웨어 제어기를 제어하는 것은 상기 운영체제에 의해 실행되는 프로세서 코어에서의 제어 태스크(서비스)에서 하드웨어 제어기에 대한 제어를 실행하는 것일 수 있다. 여기에서의 제어 태스크는 대응되는 제어 태스크를 가리킬 수 있다. 타깃 기기의 하드웨어 제어기의 경우, 제1 프로세서 코어에서 제1 운영체제의 제1 제어 트스크(제1 제어 프로세스)를 실행할 수 있으며, 제1 제어 트스크에서 하드웨어 제어기를 제어할 수 있다.
- [0472] 하드웨어 제어기를 제어하는 것은 센서의 센서 데이터에 기반하여 수행되는 것이다. 서로 다른 타깃 기기의 경우, 그의 실행에 영향주는 파라미터는 서로 다를 수 있다. 대응되게, 획득해야 할 센서 데이터에도 구별점이 존재할 수 있다. 타깃 기기의 경우, 이는 칩의 전원이 켜진 후 바로 실행되는 기기일 수 있고, 이와 대응되는 센서는 지정된 센서일 수 있다. 지정된 센서의 유형은 여러가지 존재할 수 있으며, 이는 온도 센서, 습도 센서, 잡음 센서 등을 포함하지만, 그 중의 적어도 하나에 제한되지 않는다. 제1 제어 태스크는 제1 프로세서 코어에서 실행되므로, 제1 프로세서 코어를 통해 지정된 센서의 센서 데이터를 읽을 수 있다. 지정된 센서의 센서 데이터는 지정된 센서 내의 저장 공간에 저장될 수 있으며, 지정된 센서에 의해 지정된 저장 공간 내에 전송할 수 있다. 본 실시예에서는, 지정된 센서의 센서 데이터를 읽는 위치에 대해 한정하지 않는다.
- [0473] 읽은 지정된 센서의 센서 데이터는 하나의 주기 내의 센서 데이터일 수도 있고, 타깃 기기가 시작된 후부터의 전부 센서 데이터일 수도 있으며, 다른 시간 제한 조건을 충족시키는 센서 데이터일 수도 있다. 지정된 센서의 센서 데이터를 획득한 후, 제1 제어 태스크에서는 지정된 센서의 센서 데이터에 따라 타깃 기기의 실행 상태를 제어할 수 있다. 타깃 기기의 실행 상태를 제어하는 것은 하기의 방식을 통해 실현할 수 있다. 제1 제어 태스크를 통해 타깃 기기의 하드웨어 제어기로 기기 제어 명령어를 발신하여, 하드웨어 제어기에서 기기 제어 명령어에 따라 타깃 기기의 실행 상태를 제어한다.
- [0474] 선택적으로, 제1 제어 태스크에서는 지정 센서의 센서 데이터에 따라 타깃 기기의 예상 실행 상태를 결정할 수 있다. 타깃 기기의 현재 실행 상태가 예상 실행 상태와 다를 경우, 상기 기기 제어 명령어를 생성할 수 있다. 기기 제어 명령어는 타깃 기기의 실행 상태를 예상 실행 상태로 조정하도록 제어할 수 있다. 상기 기기 제어 명령어는 제2 버스를 거쳐 타깃 기기의 하드웨어 제어기에 발신하는 것일 수 있다. 제2 버스는 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0475] 지정된 센서의 센서 데이터를 읽고, 센서 데이터에 따라 타깃 기기를 제어하여, 실행 상태를 제어함으로써, 리소스의 이용율을 향상시킨다.
- [0476] 예시적인 일 실시예에서는, 제1 제어 태스크를 통해 지정 센서의 센서 데이터에 따라 제2 버스를 거쳐 하드웨어 제어기로 기기 제어 명령어를 발신하는 단계는, 제1 제어 태스크를 통해 지정 센서의 센서 데이터에 따라 타깃 기기의 기기 실행 파라미터의 타깃 파라미터값을 결정하는 단계를 포함한다. 여기서, 기기 실행 파라미터는 타깃 기기의 실행 상태를 제어하는 파라미터이고, 제1 제어 태스크를 통해 타깃 파라미터값을 가진 기기 제어 명령어를 제2 버스를 거쳐 하드웨어 제어기에 발신한다.
- [0477] 제1 제어 태스크는 지정 센서의 센서 데이터에 따라, 타깃 기기의 예상 실행 상태를 결정할 수 있다. 예상 실행 상태는 기기 실행 파라미터의 파라미터값으로 나타낼 수 있으며, 기기 실행 파라미터는 타깃 기기의 실행 상태를 제어하는 파라미터일 수 있다. 서로 다른 유형의 기기의 경우, 이에 대응되는 기기 실행 파라미터는 서로 다를 수 있다. 예를 들면, 팬의 경우, 그에 대응되는 기기 실행 파라미터는 회전속도일 수 있고, 다른 유형의 기기의 경우, 기기 실행 파라미터는 다른 실행 파라미터일 수 있다. 예상 실행 상태는 타깃 기기의 기기 실행 파라미터의 타깃 파라미터값에 대응될 수 있다.
- [0478] 타깃 기기의 기기 실행 파라미터의 타깃 파라미터값을 결정된 후, 타깃 파라미터값을 상기 기기 제어 명령어에

서 갖도록 할 수 있다. 제1 제어 태스크를 통해 타깃 파라미터값을 갖는 기기 제어 명령어를 하드웨어 제어기에 발신한다. 하드웨어 제어기로 기기 제어 명령어를 발신하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.

- [0479] 센서 데이터에 따라 타깃 기기의 기기 실행 파라미터의 파라미터값을 결정하고, 결정된 파라미터값을 기기 제어 명령어에서 갖도록하여, 기기 제어의 정확도를 향상시킬 수 있다.
- [0480] 예시적인 일 실시예에서는, 제1 제어 태스크를 통해 지정 센서의 센서 데이터에 따라 타깃 기기의 기기 실행 파라미터의 타깃 파라미터값을 결정하는 단계는, 타깃 기기가 팬일 경우, 제1 제어 태스크를 통해 지정 센서의 센서 데이터에 따라 팬의 팬 실행 파라미터의 타깃 파라미터값을 결정하는 단계를 포함한다.
- [0481] 타깃 기기가 팬일 수 있으며, 서버 또는 다른 기기를 방열시키기 위한 팬, 즉 방열 팬일 수 있다. 이때, 기기 실행 파라미터는 팬 실행 파라미터일 수 있고, 팬 실행 파라미터는 한가지 또는 여러가지 파라미터를 포함할 수 있다. 상기 기기 실행 파라미터는 회전속도, 회전주기, 주기 변환 시간을 포함할 수 있지만, 그 중의 적어도 하나에 제한되지 않으며, 다른 실행 파라미터일 수도 있다. 본 실시예에서는 이에 대해 한정하지 않는다.
- [0482] 대응되게, 제1 제어 태스크를 통해 지정 센서의 센서 데이터에 따라 타깃 기기의 기기 실행 파라미터의 타깃 파라미터값을 결정하는 것은, 제1 제어 태스크를 통해 지정 센서의 센서 데이터에 따라 팬의 팬 실행 파라미터의 타깃 파라미터값을 결정하는 것일 수 있다. 타깃 파라미터값을 획득한 후, 제1 제어 태스크는 타깃 파라미터값을 갖는 기기 제어 명령어를 제2 버스를 거쳐 팬의 하드웨어 제어기에 발신하여, 팬의 실행 상태를 제어한다.
- [0483] 팬의 실행 상태를 제어하는 것을 통해, 시스템의 전원이 켜지거나, 시스템 재부팅되거나 또는 다른 장면에서, 신속하게 팬의 실행 상태를 제어하여, 팬 제어의 적시성을 향상시킬 수 있다.
- [0484] 예시적인 일 실시예에서는, 타깃 기기가 팬일 경우, 제1 제어 태스크를 통해 지정 센서의 센서 데이터에 따라 팬의 팬 실행 파라미터의 타깃 파라미터값을 결정하는 단계는, 타깃 기기가 팬이고, 또한 지정 센서가 온도 센서일 경우, 제1 제어 태스크를 통해 온도 센서의 센서 데이터에 따라 팬의 회전속도의 타깃 회전값을 결정하되, 팬의 회전속도는 온도 센서에 의해 검출되는 온도와 양의 상관되는 것인 단계를 포함한다.
- [0485] 타깃 기기가 팬인 장면에 있어서, 지정 센서는 온도 센서일 수 있고, 상기 온도 센서의 개수는 하나 또는 복수 개일 수 있으며, 온도 센서의 설치 위치는 수요에 따라 구성될 수 있고, 서로 다른 온도 센서는 서로 다른 위치에 설치될 수 있다. 선택적으로, 온도 센서의 센서 데이터는 온도 센서에 의해 검출되는 온도를 나타내기 위한 것이다. 이에 대해, 제1 제어 태스크에서는 온도 센서의 센서 데이터에 따라 팬의 회전속도의 타깃 회전값을 결정할 수 있다. 여기서, 팬의 회전속도는 온도 센서에 의해 검출되는 온도와 양의 상관된다.
- [0486] 온도 센서의 개수가 복수 개일 경우, 온도 센서 각각의 센서 데이터에 따라, 복수의 온도 센서에 의해 검출되는 최고 온도를 결정할 수 있다. 팬의 회전속도는 복수의 온도 센서에 의해 검출되는 최고 온도에 따라 결정되는 것일 수 있다. 복수의 온도 센서에 의해 검출되는 평균 온도에 따라 팬의 회전속도를 결정하는 것에 대해, 기기 실행의 보안성을 보장할 수 있다. 팬의 개수가 복수 개인 장면에 있어서, 팬 각각과 매칭되는 온도 센서에 의해 검출되는 최고 온도 또는 평균 온도에 기반하여, 팬 각각의 회전속도를 결정할 수도 있다.
- [0487] 예를 들면, 제1 운영체제(예를 들면, RTOS 시스템)로 CPLD, EC 칩, 맞춤형 칩 등 처리 유닛을 대체하여 팬의 회전속도를 제어할 수 있다(실시간 수행되는 BMC 팬 제어일 수 있음). 시스템의 전원을 켜를 시, 제1 프로세서 코어(예를 들면, CPU0, 제1 프로세서 코어는 하드웨어에 의해 웨이크업된 것일 수 있음)를 웨이크업할 수 있고, 제1 프로세서 코어에서 부트 로드 프로그램(예를 들면, BootRom에서의 지정 프로그램)을 실행하여, 제1 운영체제를 로드하여 시작하며, 제1 프로세서 코어에서는 온도와 관련되는 다양한 센서(sensor) 데이터를 읽고, 팬 제어(예를 들면, 팬 회전속도 제어)를 수행하며, 상기 처리 유닛을 완전히 시뮬레이션하여 팬 조정 및 제어의 기능을 완성한다. 팬의 회전속도를 제어할 시, 제1 운영체제에서는 온도 센서에 따라 PWM 값을 계산하여, 팬의 회전속도를 조정할 수 있다. 상기 방식을 통해, 제2 운영체제의 시작 과정에서, 팬의 회전속도는 제1 운영체제에 의해 제어될 수 있다.
- [0488] 예시적인 일 실시예에서는, 프로세서의 제2 프로세서 코어에서 제2 운영체제의 시작을 부트하는 단계는, 2차 프로그램 로더에서 제2 프로세서 코어를 웨이크업하기 위해, 제1 프로세서 코어를 통해 2차 프로그램 로더를 실행하는 단계; 제1 프로세서 코어에서 제2 운영체제의 시작을 부트하기 위해, 제2 프로세서 코어를 통해 제2 운영체제의 범용 부트 로더를 실행하는 단계;를 포함한다.
- [0489] 본 실시형태에서는, 운영체제를 시작할 시, 2차 프로그램 로더를, 예를 들면, SOC 내부의 정적 랜덤 액세스 메

모리(Static RAM, SRAM)와 같은 내부 메모리에 로드할 수 있다. SPL은 범용 부트 로드 프로그램(Universal Boot Loader, U-Boot로 약칭함)을 랜덤 액세스 메모리에 로드하는 것을 담당할 수 있다. 2차 프로그램 로더는 제2 운영체제를 부트 로드할 수도 있고, 제1 운영체제를 부트 로드할 수도 있다.

- [0490] 제2 운영체제의 경우, 2차 프로그램 로더에서 제2 프로세서 코어를 웨이크업하기 위해, 제1 프로세서 코어를 통해 2차 프로그램 로더를 실행할 수 있고, 제1 프로세서 코어에서 제2 운영체제의 시작을 부트하기 위해, 제2 프로세서 코어를 통해 제2 운영체제의 범용 부트 로더(범용 부트 로드 프로그램)를 실행할 수 있다. 여기서, 2차 프로그램 로더를 통해 제2 운영체제의 부트 프로그램을 부트 로드하고, 제2 운영체제의 부트 프로그램은 범용 부트 로더를 포함할 수 있다.
- [0491] 설명해야 할 것은, 2차 프로그램 로더는 범용 부트 로드 프로그램 제1 단계에서 실행되는 코드로서, 범용 부트 로드 프로그램 제2 단계의 코드를 시스템 메모리(System RAM, 오프 칩 메모리라고도 함)에 옮겨 실행하도록 하는 것을 담당할 수 있다. 범용 부트 로드 프로그램은 하나의 GPL(General Public License, 범용 공공 허가 프로토콜) 프로토콜을 따르는 오픈소스 소프트웨어로서, 하나의 베어메탈 통합 루틴으로 간주할 수 있다.
- [0492] 예를 들면, 시스템의 전원을 켜 후, RTOS 시스템에서 가능한 신속하게 실행될 수 있도록, 프로세서에서는 먼저 CPU0 코어를 웨이크업하고, 다음 Bootrom에서의 프로그램을 이용하여 RTOS 시스템의 시작을 부트하며, RTOS 시스템이 시작되는 과정에서, 계속 SPL을 통해 U-Boot를 로드하고, Linux 시스템이 정상으로 시작될 때까지, U-Boot를 통해 CPU1에서 제2 운영체제의 시작을 부트한다.
- [0493] 설명해야 할 것은, Bootrom은 칩(예를 들면, SOC 칩) 내부 ROM 고정된 프로그램이며, 이는 uboot의 부트 코드이다. Bootrom에서 하드웨어의 시작 정보(예를 들면, 디프 스위치 설정)를 읽고, 지정 시작 매체(예를 들면, SD, MMC 등)으로부터 uboot-spl 코드(즉, SPL)를 읽는다. SPL은 주로 외부 RAM 및 환경을 초기화하고, 진정한 uboot 이미지를 외부 RAM에 로드하여 실행시키는 것을 담당한다. 외부 RAM은 DDR(Double Data Rate Synchronous Dynamic RAM, 더블 속도의 동기화 동적 랜덤 액세스 메모리)일 수도 있고, 다른 RAM일 수도 있다.
- [0494] 2차 프로그램 로더를 통해 제2 프로세서 코어를 웨이크업하고, 제2 프로세서 코어에서 범용 부트 로드 프로그램을 실행하여, 대응되는 프로세서 코어에서의 제2 운영체제를 부트함으로써, 운영체제 시작의 편의성 및 성공율을 향상시킬 수 있다.
- [0495] 선택적인 일 예시로서, 아래는 RTOS 시스템 및 Linux 시스템을 예로 들어 멀티 코어 듀얼 시스템의 시작 과정을 해석하여 설명한다.
- [0496] 가능한 신속하게 팬 관리를 연결 관리하기 위해, 가능한 RTOS 시스템이 시작될 수 있도록 하고, Linux 시스템 시작을 완성한 후, Linux 시스템에서 팬의 제어 권한을 연결 관리한다. 멀티 코어 듀얼 시스템의 시작 과정은 하기와 같은 단계 1, 단계 2, 단계 3, 단계 4, 단계 5 및 단계 6을 포함할 수 있다.
- [0497] 단계 1에서, 시스템의 전원을 켜는 시, CPU0을 웨이크업한다.
- [0498] 단계 2에서, CPU0은 Bootrom에서의 지정 프로그램을 실행하고, RTOS 시스템의 시작을 로드한다.
- [0499] 단계 3에서, RTOS 시스템이 시작되는 과정에서, CPU1을 웨이크업하여 u-boot를 부트하며, 제1 운영체제에서의 팬 제어 프로그램(FanCtrl_RTOS_APP)을 시작한다.
- [0500] 단계 4에서, CPU1에서 u-boot에 대한 부트는 SPL 단계 및 Uboot 단계를 포함할 수 있으며, SPL을 호출하여 SPL 단계에 진입한다.
- [0501] 단계 5에서, SPL 단계에서, SPL은 Uboot의 시작을 부트한다.
- [0502] 단계 6에서, Uboot 단계에서, Linux 코어(CPU1 내지 CPUN)를 로드하며, BMC 서비스 프로그램 및 제2 운영체제에서의 팬 제어 프로그램(FanCtrl_Linux_APP)을 시작한다.
- [0503] 본 선택적인 예시를 통해, 듀얼 시스템이 시작 및 실행되는 과정에서, 먼저 RTOS 시스템을 시작하여 팬을 제어하고, Linux 시스템이 시작된 후, 제2 운영체제에서 팬의 제어 권한을 연결 관리하므로, 시스템의 전원이 켜질 시 신속하게 팬을 제어하도록 보장할 수 있고, 팬 제어의 효율을 향상시킬 수 있다.
- [0504] 예시적인 일 실시예에서는, 제2 운영체제를 통해 제2 버스를 거쳐 하드웨어 제어기를 연결 관리하는 단계 이후, 제2 운영체제가 재부팅될 경우, 제2 운영체제를 통해 제1 버스를 거쳐 제1 운영체제를 웨이크업하며, 제1 운영체제에서 제2 버스를 거쳐 하드웨어 제어기를 연결 관리하여, 타겟 기기의 제어 권한을 연결 관리하는 단계; 제

2 운영체제를 제어하여 시스템을 재부팅하는 단계; 를 더 포함한다.

- [0505] 시스템 크래시, reboot(재부팅) 명령 수신 등 이유로 재부팅해야 할 경우, 제2 운영체제에서는 먼저 제1 운영체제를 웨이크업하고, 제1 운영체제에서 하드웨어 제어기를 연결 관리하여, 타깃 기기의 제어 권한을 연결 관리할 수 있다. 제1 운영체제를 웨이크업하는 것은 제1 버스를 거쳐 실행될 수 있으며, 제1 운영체제에서 하드웨어 제어기를 연결 관리하는 것은 제2 버스를 거쳐 실행될 수 있다.
- [0506] 제2 운영체제가 재부팅될 경우, 제1 운영체제를 웨이크업하여 타깃 기기의 제어 권한을 연결 관리하는 단계를 통해, 기기 제어의 신뢰도를 향상시킬 수 있다.
- [0507] 예시적인 일 실시예에서는, 제2 운영체제가 재부팅될 경우, 제2 운영체제를 통해 제1 버스를 거쳐 제1 운영체제를 웨이크업하는 단계는, 제2 운영체제가 재부팅될 경우, 제2 운영체제를 통해 제1 버스를 거쳐 제1 운영체제로 시스템 웨이크업 중단을 발신하되, 시스템 웨이크업 중단은 제1 운영체제를 웨이크업하기 위한 것인 단계를 포함한다.
- [0508] 제1 운영체제에 대한 웨이크업은 코어 간 중단을 통해 실현될 수 있다. 제2 운영체제가 재부팅될 경우(예를 들면, 시스템 크래시, reboot 명령 수신), 제2 운영체제에서는 제1 운영체제로 시스템 웨이크업 중단을 발신하여, 제1 운영체제를 웨이크업할 수 있다. 상기 시스템 웨이크업 중단은 능동적인 웨이크업 중단일 수 있다. 제1 운영체제에서 하드웨어 제어기를 연결 관리한 후, 제2 운영체제를 제어하여 시스템을 재부팅할 수 있다. 제2 운영체제가 재부팅된 후, 하드웨어 제어기를 다시 연결 관리할 수 있다. 하드웨어 제어기를 연결 관리하는 과정은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0509] 상기 실시형태의 설명을 통해, 당업자는 상기 실시예에 따른 방법은 소프트웨어 및 필요한 범용 하드웨어 플랫폼을 통해 실현될 수 있으며, 물론 하드웨어에 의해 실현될 수도 있지만, 많은 경우에는 전자가 더 바람직한 실시형태임을 명확하게 이해할 수 있다. 이러한 이해에 기초하여, 본 출원의 해결수단의 본질 또는 관련 기술에 기여한 부분은 소프트웨어 제품의 형식으로 구현될 수 있으며, 상기 컴퓨터 소프트웨어 제품은 하나의 저장 매체(예를 들면, ROM/RAM, 자기 디스크, 광 디스크)에 저장되며, 단말기 기기(핸드폰, 컴퓨터, 서버 또는 네트워크 디바이스 등일 수 있음)가 본 출원의 각 실시예의 방법을 수행하도록 하는 여러 명령을 포함한다.
- [0510] 본 출원의 실시예의 다른 일 측면에 따르면, 상기 임베디드 시스템의 실행 방법을 실시하기 위한 임베디드 시스템을 더 제공한다. 상기 임베디드 시스템은 상기 BMC 칩에서 실행될 수 있다. 여기서, 상기 임베디드 시스템은 제1 운영체제, 제2 운영체제, 서비스 관리 모듈, 리소스 동적 할당 모듈 및 리소스 적응형 스케줄링 모듈을 포함할 수 있다.
- [0511] 제1 운영체제 및 제2 운영체제는, 프로세서에서 실행된다. 제1 운영체제의 응답 속도는 제2 운영체제보다 빠르다.
- [0512] 서비스 관리 모듈은, 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당한다. 여기서, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당을 수행한다.
- [0513] 리소스 동적 할당 모듈은, 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정한다. 여기서, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함한다.
- [0514] 리소스 적응형 스케줄링 모듈은, 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.
- [0515] 본 실시예에서는, 제1 운영체제 및 제2 운영체제는 상기 실시예와 유사하며, 여기에서 설명을 생략한다. 서비스 관리 모듈, 리소스 동적 할당 모듈 및 리소스 적응형 스케줄링 모듈은 제1 운영체제 및 제2 운영체제에서의 소프트웨어 모듈에서 실행될 수 있다. 상기 모듈을 구분하여, 서로 다른 기능의 모듈에 대한 개발과 유지보수 수행이 용이할 수 있고, 동시에, 리소스 동적 할당 규칙의 경우, 리소스 동적 할당 규칙을 원활하게 설정하여, 리소스 할당의 원활성을 향상시킬 수 있다.
- [0516] 상기 임베디드 시스템에 있어서, 상기 임베디드 시스템은, 프로세서에서 실행되되, 제1 운영체제의 응답 속도가 제2 운영체제보다 빠른 제1 운영체제 및 제2 운영체제; 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하되, 상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것인 서비스 관리 모듈; 한세트의 할당 대상인 서비

스와 대응되는 리소스 할당 결과를 결정한다, 상기 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것인 리소스 동적 할당 모듈; 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 리소스 적응형 스케줄링 모듈; 을 포함하는 것을 통해, 관련 기술에 존재하는 멀티 코어 프로세서 다수의 처리 리소스가 유휴 상태인 것으로 인한 코어 리소스의 전반 이용률이 낮은 과제를 해결함으로써, 처리 리소스의 이용률을 향상시킨다.

- [0517] 예시적인 일 실시예에서는, 임베디드 시스템은, 로드 밸런스 전략 모듈을 더 포함한다.
- [0518] 로드 밸런스 전략 모듈은, 규칙 구성 파일을 읽고, 규칙 구조체를 생성한다. 여기서, 규칙 구조체는 리소스 동적 할당 규칙을 기록하기 위한 것이다.
- [0519] 본 실시예에서의 로드 밸런스 전략 모듈은 상기 실시예와 유사하며, 여기에서 설명을 생략한다. 여기서, 로드 밸런스 전략 모듈은 제1 운영체제 또는 제2 운영체제에서의 소프트웨어 모듈일 수 있으며, 로드 밸런스 전략을 별도의 소프트웨어 모듈 세트로 구성하여 저장하는 것을 통해, 로드 밸런스 전략에 대한 원활한 조정이 용이할 수 있다.
- [0520] 예시적인 일 실시예에서는, 로드 밸런스 전략 모듈은 또한,
- [0521] 제2 운영체제의 외부 인터페이스를 통해 규칙 업데이트 구성 파일을 획득하되, 규칙 업데이트 구성 파일은 구성된 리소스 동적 할당 규칙을 업데이트하고, 규칙 업데이트 구성 파일을 사용하여 규칙 구조체를 업데이트하여, 규칙 구조체에 기록된 리소스 동적 할당 규칙을 업데이트한다.
- [0522] 본 실시예에서는, 로드 밸런스 전략 모듈에 저장된 리소스 동적 할당 규칙은 제2 운영체제의 외부 인터페이스를 통해 원활하게 구성될 수 있고, 구성 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0523] 예시적인 일 실시예에서는, 서비스 관리 모듈은 하기와 같은 단계를 실행하여 리소스 동적 할당 규칙에서 서비스 응답 속도와 대응되는 할당 규칙에 따라 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0524] 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0525] 본 실시예에서는, 서비스 관리 모듈이 서비스 응답 속도에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0526] 예시적인 일 실시예에서는, 서비스 관리 모듈은 하기와 같은 단계를 실행하여 리소스 동적 할당 규칙에서 서비스 서비스 리소스 점유율과 대응되는 할당 규칙에 따라 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0527] 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0528] 본 실시예에서는, 서비스 관리 모듈이 서비스 리소스 점유율에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0529] 예시적인 일 실시예에서는, 서비스 관리 모듈은 또한 하기와 같은 단계 중의 적어도 한 단계를 실행하여 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0530] 한세트의 할당 대상인 서비스에서 제1 운영체제의 할당된 서비스와의 서비스 결합도가 제1 결합도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당한다.
- [0531] 한세트의 할당 대상인 서비스에서 제2 운영체제의 할당된 서비스와의 서비스 결합도가 제2 결합도 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0532] 본 실시예에서는, 서비스 관리 모듈이 서비스 결합도에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.

- [0533] 예시적인 일 실시예에서는, 서비스 관리 모듈은 또한 하기와 같은 단계를 실행하여 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0534] 한세트의 할당 대상인 서비스에 민감한 정보가 포함되는 할당 대상인 서비스를 타깃 운영체제에 할당한다. 여기서, 타깃 운영체제는 제1 운영체제 및 제2 운영체제에서 사용 객체와 인터랙션하는 빈도가 낮은 운영체제이다.
- [0535] 본 실시예에서는, 서비스 관리 모듈이 민감한 정보를 포함하는 할당 대상인 서비스를 제1 운영체제에 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0536] 설명해야 할 것은, 서비스 관리 모듈에서 서비스 응답 속도 및 서비스 리소스 점유율에 기반하여 할당 대상인 서비스를 할당하는 것은 로드 밸런싱 전략 모듈에 저장된 리소스 동적 할당 규칙에 기반하여 실행되는 것이고, 서비스 결합도 및 서비스 중요성(예를 들면, 민감한 정보가 포함되는지 여부)에 기반하여 할당 대상인 서비스를 할당하는 것은 서비스 관리 모듈에서 그에 사전 설정된 구성 정보에 기반하여 실행되는 것일 수 있으며, 서비스 결합도 및 서비스 중요성에 기반하는 처리 리소스 할당 규칙은 통상적으로 안정적이고, 서비스 응답 속도 및 서비스 리소스 점유율은 수요에 따라 원활하게 구성될 수 있으며, 별도의 소프트웨어 모듈을 통해 서비스 응답 속도 및 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당되는 리소스 동적 할당 규칙을 저장하므로, 규칙 구성의 원활성과 간결성을 모두 고려할 수 있다.
- [0537] 예시적인 일 실시예에서는, 리소스 동적 할당 모듈은 하기와 같은 단계를 실행하여 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 것을 실현한다.
- [0538] 서비스 관리 모듈의 할당 결과에 따라, 제1 운영체제의 처리 리소스의 리소스 이용 상황 및 제2 운영체제의 처리 리소스의 리소스 이용 상황을 결합하여, 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성한다.
- [0539] 본 실시예에서는, 리소스 동적 할당 모듈에서 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0540] 예시적인 일 실시예에서는, 리소스 동적 할당 모듈은 프로세서 코어를 단위로 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.
- [0541] 예시적인 일 실시예에서는, 리소스 적응형 스케줄링 모듈은 하기와 같은 단계를 실행하여 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 것을 실현한다.
- [0542] 리소스 할당 결과에 따라 프로세서의 처리 리소스에서의 할당되지 않은 처리 리소스에는 대응되는 할당 대상인 서비스가 존재하는 것으로 결정할 경우, 할당되지 않은 처리 리소스를 할당되지 않은 처리 리소스와 대응되는 할당 대상인 서비스에 의해 할당되는 운영체제에 할당한다.
- [0543] 본 실시예에서는, 리소스 적응형 스케줄링 모듈에서 할당되지 않은 처리 리소스를 운영체제에 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0544] 예시적인 일 실시예에서는, 임베디드 시스템은 리소스 선점과 해제 모듈을 더 포함한다.
- [0545] 리소스 선점과 해제 모듈은, 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0546] 본 실시예에서는, 리소스 선점과 해제 모듈이 서로 다른 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0547] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0548] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것을 실현한다.
- [0549] 코어 간 통신 인터페이스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송하되, 제1 인터랙션 요청은 제2 운영체제와 리소스 인터랙션을 수행하는 것을 요청하기 위한 것이고, 리소스 인터랙션은 리소스 선점 및 리소스 해제 중의 하나를 포함한다.
- [0550] 코어 간 통신 인터페이스를 통해 제2 운영체제에서 제1 인터랙션 요청에 응답하여 반환하는 제1 인터랙션 응답

을 획득하되, 제1 인터랙션 응답은 제1 운영체제에서 제1 인터랙션 응답에 따라 제2 운영체제와 리소스 인터랙션을 수행하는 것을 지시하기 위한 것이다.

- [0551] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0552] 예시적인 일 실시예에서는, 임베디드 시스템은 제1 시스템 제어 모듈을 더 포함한다.
- [0553] 제1 시스템 제어 모듈은, 제1 운영체제의 처리 리소스의 리소스 이용율을 검출하되, 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 제1 운영체제에서 제2 운영체제와 리소스 인터랙션을 수행하는 것을 대기하는 것으로 결정할 경우, 리소스 선점과 해제 모듈에서 코어 간 통신 인터랙션을 통해 제1 인터랙션 요청을 제2 운영체제에 전송하는 것을 트리거한다.
- [0554] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제의 처리 리소스의 처리 이용율에 기반하여 리소스 인터랙션을 트리거하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0555] 예시적인 일 실시예에서는, 제1 시스템 제어 모듈은, 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 제1 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정하는 단계; 제1 운영체제의 처리 리소스가 유휴 상태이고, 제1 운영체제에서 실행 대상인 서비스가 존재하지 않을 경우, 제1 운영체제에서 제2 운영체제로 처리 리소스를 해제하는 것을 대기하는 것으로 결정하는 단계; 중의 적어도 한 단계를 실행하기도 하다.
- [0556] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제에서 수행할 리소스 인터랙션 유형을 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0557] 예시적인 일 실시예에서는, 제1 시스템 제어 모듈은 또한, 제1 운영체제에 스케줄링할 서비스가 없고, 제1 운영체제에 할당 대상인 서비스가 없을 경우, 제1 운영체제가 휴면 상태에 진입하도록 제어한다.
- [0558] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제에서 휴면 상태에 진입하도록 제어하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0559] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하여 코어 간 통신 인터페이스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송하는 것을 실현한다.
- [0560] 코어 간 통신 인터페이스를 통해 중단 번호가 제1 중단 번호인 제1 선점 요청을 제2 운영체제에 전송하되, 제1 선점 요청은 제2 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0561] 코어 간 통신 인터페이스를 통해 중단 번호가 제2 중단 번호인 리소스 해제 요청을 제2 운영체제에 전송하되, 리소스 해제 요청은 제2 운영체제에 제1 운영체제에 의해 점유된 처리 리소스를 해제하는 것을 요청하기 위한 것이다.
- [0562] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 중단을 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0563] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 코어 간 통신 인터페이스를 통해 제1 운영체제와 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것을 실현한다.
- [0564] 코어 간 통신 인터페이스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송하되, 제2 인터랙션 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0565] 코어 간 통신 인터페이스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하되, 제2 인터랙션 응답은 제1 운영체제가 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는지 여부를 지시하기 위한 것이다.
- [0566] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0567] 예시적인 일 실시예에서는, 임베디드 시스템은 제2 시스템 제어 모듈을 더 포함한다.
- [0568] 제2 시스템 제어 모듈은, 제2 운영체제의 처리 리소스의 리소스 이용율을 검출하되, 제2 운영체제의 처리 리소스의 리소스 이용율에 따라, 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로

결정할 경우, 리소스 선점과 해제 모듈에서 코어 간 통신 인터페이스를 통해 제2 인터랙션 요청을 제1 운영체제에 전송하도록 트리거한다.

- [0569] 본 실시예에서는, 제2 시스템 제어 모듈이 제2 운영체제의 처리 리소스에 기반하여 리소스 인터랙션을 트리거하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0570] 예시적인 일 실시예에서는, 제2 시스템 제어 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하기도 하다.
- [0571] 제2 운영체제의 처리 리소스의 현재의 리소스 이용율이 제2 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0572] 제2 운영체제의 실행 중 서비스 및 제2 운영체제의 실행 대상인 서비스에 따라 제2 운영체제의 처리 리소스의 리소스 이용율이 제3 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0573] 본 실시예에서는, 제2 시스템 제어 모듈이 제2 운영체제에서 수행될 리소스 인터랙션을 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0574] 예시적인 일 실시예에서는, 제1 운영체제는 또한, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮지 않을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것으로 결정하고, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것으로 결정한다.
- [0575] 본 실시예에서는, 임베디드 시스템이 제1 운영체제에서 제2 운영체제가 처리 리소스를 점유하는 것을 허용하는지 여부를 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0576] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 코어 간 통신 인터페이스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송하는 것을 실현한다.
- [0577] 코어 간 통신 인터페이스를 통해 중단 번호가 제3 중단 번호인 제2 선점 요청을 제1 운영체제에 전송하되, 제2 선점 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0578] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 중단을 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0579] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하여 코어 간 통신 인터페이스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하는 것을 실현한다.
- [0580] 코어 간 통신 인터페이스를 통해 중단 번호가 제4 중단 번호인 리소스 해제를 허용하는 응답을 제2 운영체제에 전송하되, 리소스 해제를 허용하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것을 지시하기 위한 것이다.
- [0581] 코어 간 통신 인터페이스를 통해 중단 번호가 제5 중단 번호인 리소스 해제를 거부하는 응답을 제2 운영체제에 전송하되, 리소스 해제를 거부하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것을 지시하기 위한 것이다.
- [0582] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 중단을 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0583] 예시적인 일 실시예에서는, 리소스 적응형 스케줄링 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하기도 하다. 제1 운영체제에 의해 선점된 제2 운영체제의 처리 리소스를 제1 운영체제의 사용 가능한 리소스풀에 병합하고, 제2 운영체제에 의해 선점된 제1 운영체제의 처리 리소스 또는 제1 운영체제에서 제2 운영체제에 능동적으로 해제한 처리 리소스를 제2 운영체제의 사용 가능한 리소스풀에 병합한다.
- [0584] 본 실시예에서는, 리소스 적응형 스케줄링 모듈이 서로 다른 운영체제 사이에서 처리 리소스 스케줄링을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0585] 예시적인 일 실시예에서는, 임베디드 시스템은 시스템 시작 모듈을 더 포함한다.

- [0586] 시스템 시작 모듈은, 프로세서가 위치하는 칩의 전원이 켜진 후, 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하고, 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제의 실행을 부트하되, 제1 초기 처리 리소스는 프로세서의 처리 리소스에서 제1 운영체제와 대응되는 초기 처리 리소스이고, 제2 초기 처리 리소스는 프로세서의 처리 리소스에서 제2 운영체제와 대응되는 초기 처리 리소스이다.
- [0587] 본 실시예에서는, 시스템 시작 모듈이 부트 로드 프로그램을 통해 대응되는 초기 처리 리소스에서 각 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0588] 예시적인 일 실시예에서는, 제1 초기 처리 리소스는 프로세서의 프로세서 코어에서의 하나의 지정된 프로세서 코어이고, 제2 초기 처리 리소스는 프로세서의 프로세서 코어에서 지정된 프로세서 코어와의 다른 프로세서 코어이다.
- [0589] 예시적인 일 실시예에서는, 시스템 시작 모듈은 하기와 같은 단계를 실행하여 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하는 것을 실현한다.
- [0590] 2차 프로그램 로더를 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트한다.
- [0591] 본 실시예에서는, 시스템 시작 모듈이 2차 프로그램 로더를 통해 초기 처리 리소스에서 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0592] 예시적인 일 실시예에서는, 시스템 시작 모듈은, 기기의 메인보드에 고정된 시작 칩에서의 사전 설정된 프로그램을 통해 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행하되, 2차 프로그램 로더는 보안 시작 점검의 점검 결과가 정상인 후에 실행된다.
- [0593] 본 실시예에서는, 시스템 시작 모듈이 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0594] 예시적인 일 실시예에서는, 시스템 시작 모듈은 하기와 같은 단계를 실행하여 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제의 실행을 부트하는 것을 실현한다.
- [0595] 2차 프로그램 로더를 통해 제2 초기 처리 리소스를 웨이크업하고, 범용의 부트 로드 프로그램의 실행을 부트하여, 제2 운영체제의 실행을 부트한다.
- [0596] 본 실시예에서는, 시스템 시작 모듈이 2차 프로그램 로더를 통해 제2 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0597] 본 출원의 실시예의 다른 일 측면에 따르면, 상기 임베디드 시스템의 실행 방법을 실시하기 위한 임베디드 시스템을 더 제공한다. 상기 임베디드 시스템은 상기 BMC 칩에서 실행될 수 있다. 여기서, 상기 임베디드 시스템은 제1 운영체제, 제2 운영체제, 서비스 관리 모듈, 리소스 동적 할당 모듈 및 리소스 적응형 스케줄링 모듈을 포함할 수 있다.
- [0598] 제1 운영체제 및 제2 운영체제는, 프로세서에서 실행된다. 제1 운영체제의 응답 속도는 제2 운영체제보다 빠르다.
- [0599] 서비스 관리 모듈은, 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당한다. 여기서, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당을 수행한다.
- [0600] 리소스 동적 할당 모듈은, 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정한다. 여기서, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함한다.
- [0601] 리소스 적응형 스케줄링 모듈은, 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.
- [0602] 본 실시예에서는, 제1 운영체제 및 제2 운영체제는 상기 실시예와 유사하며, 여기에서 설명을 생략한다. 서비스 관리 모듈, 리소스 동적 할당 모듈 및 리소스 적응형 스케줄링 모듈은 제1 운영체제 및 제2 운영체제에서의 소프트웨어 모듈에서 실행될 수 있다. 상기 모듈을 구분하여, 서로 다른 기능의 모듈에 대한 개발과 유지보수 수행이 용이할 수 있고, 동시에, 리소스 동적 할당 규칙의 경우, 리소스 동적 할당 규칙을 원활하게 설정하여, 리

소스 할당의 원활성을 향상시킬 수 있다.

- [0603] 상기 임베디드 시스템에 있어서, 상기 임베디드 시스템은, 프로세서에서 실행되되, 제1 운영체제의 응답 속도가 제2 운영체제보다 빠른 제1 운영체제 및 제2 운영체제; 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하되, 상기 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것인 서비스 관리 모듈; 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하되, 상기 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것인 리소스 동적 할당 모듈; 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 리소스 적응형 스케줄링 모듈; 을 포함하는 것을 통해, 관련 기술에 존재하는 멀티 코어 프로세서 다수의 처리 리소스가 유향 상태인 것으로 인한 코어 리소스의 전반 이용률이 낮은 과제를 해결함으로써, 처리 리소스의 이용률을 향상시킨다.
- [0604] 예시적인 일 실시예에서는, 임베디드 시스템은, 로드 밸런스 전략 모듈을 더 포함한다.
- [0605] 로드 밸런스 전략 모듈은, 리소스 동적 할당 규칙을 기록하기 위한 규칙 구성 파일을 읽고, 규칙 구조체를 생성한다.
- [0606] 본 실시예에서의 로드 밸런스 전략 모듈은 상기 실시예와 유사하며, 여기에서 설명을 생략한다. 여기서, 로드 밸런스 전략 모듈은 제1 운영체제 또는 제2 운영체제에서의 소프트웨어 모듈일 수 있으며, 로드 밸런스 전략을 별도의 소프트웨어 모듈 세트로 구성하여 저장하는 것을 통해, 로드 밸런스 전략에 대한 원활한 조정이 용이할 수 있다.
- [0607] 예시적인 일 실시예에서는, 로드 밸런스 전략 모듈은 또한,
- [0608] 제2 운영체제의 외부 인터페이스를 통해 규칙 업데이트 구성 파일을 획득하되, 규칙 업데이트 구성 파일은 구성된 리소스 동적 할당 규칙을 업데이트하기 위한 것이고, 규칙 업데이트 구성 파일을 사용하여 규칙 구조체를 업데이트하여, 규칙 구조체에 기록된 리소스 동적 할당 규칙을 업데이트한다.
- [0609] 본 실시예에서는, 로드 밸런스 전략 모듈에 저장된 리소스 동적 할당 규칙은 제2 운영체제의 외부 인터페이스를 통해 원활하게 구성될 수 있고, 구성 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0610] 예시적인 일 실시예에서는, 서비스 관리 모듈은 하기와 같은 단계를 실행하여 리소스 동적 할당 규칙에서 서비스 응답 속도와 대응되는 할당 규칙에 따라 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0611] 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0612] 본 실시예에서는, 서비스 관리 모듈이 서비스 응답 속도에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0613] 예시적인 일 실시예에서는, 서비스 관리 모듈은 하기와 같은 단계를 실행하여 리소스 동적 할당 규칙에서 서비스 리소스 점유율과 대응되는 할당 규칙에 따라 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0614] 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0615] 본 실시예에서는, 서비스 관리 모듈이 서비스 리소스 점유율에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0616] 예시적인 일 실시예에서는, 서비스 관리 모듈은 또한 하기와 같은 단계 중의 적어도 한 단계를 실행하여 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0617] 한세트의 할당 대상인 서비스에서 제1 운영체제의 할당된 서비스와의 서비스 결합도가 제1 결합도 임계값 이상

인 할당 대상인 서비스를 제1 운영체제에 할당한다.

- [0618] 한세트의 할당 대상인 서비스에서 제2 운영체제의 할당된 서비스와의 서비스 결합도가 제2 결합도 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0619] 본 실시예에서는, 서비스 관리 모듈이 서비스 결합도에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0620] 예시적인 일 실시예에서는, 서비스 관리 모듈은 또한 하기와 같은 단계를 실행하여 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0621] 한세트의 할당 대상인 서비스에 민감한 정보가 포함되는 할당 대상인 서비스를 타깃 운영체제에 할당한다. 여기서, 타깃 운영체제는 제1 운영체제 및 제2 운영체제에서 사용 객체와 인터랙션하는 빈도가 낮은 운영체제이다.
- [0622] 본 실시예에서는, 서비스 관리 모듈이 민감한 정보를 포함하는 할당 대상인 서비스를 제1 운영체제에 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0623] 설명해야 할 것은, 서비스 관리 모듈에서 서비스 응답 속도 및 서비스 리소스 점유율에 기반하여 할당 대상인 서비스를 할당하는 것은 로드 밸런싱 전략 모듈에 저장된 리소스 동적 할당 규칙에 기반하여 실행되는 것이고, 서비스 결합도 및 서비스 중요성(예를 들면, 민감한 정보가 포함되는지 여부)에 기반하여 할당 대상인 서비스를 할당하는 것은 서비스 관리 모듈에서 그에 사전 설정된 구성 정보에 기반하여 실행되는 것일 수 있으며, 서비스 결합도 및 서비스 중요성에 기반하는 처리 리소스 할당 규칙은 통상적으로 안정적이고, 서비스 응답 속도 및 서비스 리소스 점유율은 수요에 따라 원활하게 구성될 수 있으며, 별도의 소프트웨어 모듈을 통해 서비스 응답 속도 및 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당되는 리소스 동적 할당 규칙을 저장하므로, 규칙 구성의 원활성과 간결성을 모두 고려할 수 있다.
- [0624] 예시적인 일 실시예에서는, 리소스 동적 할당 모듈은 하기와 같은 단계를 실행하여 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하는 것을 실현한다.
- [0625] 서비스 관리 모듈의 할당 결과에 따라, 제1 운영체제의 처리 리소스의 리소스 이용 상황 및 제2 운영체제의 처리 리소스의 리소스 이용 상황을 결합하여, 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성한다.
- [0626] 본 실시예에서는, 리소스 동적 할당 모듈에서 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0627] 예시적인 일 실시예에서는, 리소스 동적 할당 모듈은 프로세서 코어를 단위로 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.
- [0628] 예시적인 일 실시예에서는, 리소스 적응형 스케줄링 모듈은 하기와 같은 단계를 실행하여 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 것을 실현한다.
- [0629] 리소스 할당 결과에 따라 프로세서의 처리 리소스에서의 할당되지 않은 처리 리소스에는 대응되는 할당 대상인 서비스가 존재하는 것으로 결정할 경우, 할당되지 않은 처리 리소스를 할당되지 않은 처리 리소스와 대응되는 할당 대상인 서비스에 의해 할당되는 운영체제에 할당한다.
- [0630] 본 실시예에서는, 리소스 적응형 스케줄링 모듈에서 할당되지 않은 처리 리소스를 운영체제에 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0631] 예시적인 일 실시예에서는, 임베디드 시스템은 리소스 선점과 해제 모듈을 더 포함한다.
- [0632] 리소스 선점과 해제 모듈은, 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0633] 본 실시예에서는, 리소스 선점과 해제 모듈이 서로 다른 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0634] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0635] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 코어 간 통신 인터페이스

를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것을 실현한다.

- [0636] 코어 간 통신 인터페이스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송하되, 제1 인터랙션 요청은 제2 운영체제와 리소스 인터랙션을 수행하는 것을 요청하기 위한 것이고, 리소스 인터랙션은 리소스 선점 및 리소스 해제 중의 하나를 포함한다.
- [0637] 코어 간 통신 인터페이스를 통해 제2 운영체제에서 제1 인터랙션 요청에 응답하여 반환하는 제1 인터랙션 응답을 획득하되, 제1 인터랙션 응답은 제1 운영체제에서 제1 인터랙션 응답에 따라 제2 운영체제와 리소스 인터랙션을 수행하는 것을 지시하기 위한 것이다.
- [0638] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0639] 예시적인 일 실시예에서는, 임베디드 시스템은 제1 시스템 제어 모듈을 더 포함한다.
- [0640] 제1 시스템 제어 모듈은, 제1 운영체제의 처리 리소스의 리소스 이용율을 검출하되, 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 제1 운영체제에서 제2 운영체제와 리소스 인터랙션을 수행하는 것을 대기하는 것으로 결정할 경우, 리소스 선점과 해제 모듈에서 코어 간 통신 인터랙션을 통해 제1 인터랙션 요청을 제2 운영체제에 전송하는 것을 트리거한다.
- [0641] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제의 처리 리소스의 처리 이용율에 기반하여 리소스 인터랙션을 트리거하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0642] 예시적인 일 실시예에서는, 제1 시스템 제어 모듈은, 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 제1 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정하는 단계; 제1 운영체제의 처리 리소스가 유휴 상태이고, 제1 운영체제에서 실행 대상인 서비스가 존재하지 않을 경우, 제1 운영체제에서 제2 운영체제로 처리 리소스를 해제하는 것을 대기하는 것으로 결정하는 단계; 중의 적어도 한 단계를 실행하기도 하다.
- [0643] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제에서 수행할 리소스 인터랙션 유형을 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0644] 예시적인 일 실시예에서는, 제1 시스템 제어 모듈은 또한, 제1 운영체제에 스케줄링할 서비스가 없고, 제1 운영체제에 할당 대상인 서비스가 없을 경우, 제1 운영체제가 휴면 상태에 진입하도록 제어한다.
- [0645] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제에서 휴면 상태에 진입하도록 제어하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0646] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 코어 간 통신 인터페이스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송하는 것을 실현한다.
- [0647] 코어 간 통신 인터페이스를 통해 중단 번호가 제1 중단 번호인 제1 선점 요청을 제2 운영체제에 전송하되, 제1 선점 요청은 제2 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0648] 코어 간 통신 인터페이스를 통해 중단 번호가 제2 중단 번호인 리소스 해제 요청을 제2 운영체제에 전송하되, 리소스 해제 요청은 제2 운영체제에 제1 운영체제에 의해 점유된 처리 리소스를 해제하는 것을 요청하기 위한 것이다.
- [0649] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 중단을 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0650] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 코어 간 통신 인터페이스를 통해 제1 운영체제와 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것을 실현한다.
- [0651] 코어 간 통신 인터페이스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송하되, 제2 인터랙션 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0652] 코어 간 통신 인터페이스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하되, 제2 인터랙션 응답은 제1 운영체제가 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는지 여부를 지시하기 위한 것이다.

- [0653] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0654] 예시적인 일 실시예에서는, 임베디드 시스템은 제2 시스템 제어 모듈을 더 포함한다.
- [0655] 제2 시스템 제어 모듈은, 제2 운영체제의 처리 리소스의 리소스 이용율을 검출하되, 제2 운영체제의 처리 리소스의 리소스 이용율에 따라, 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정할 경우, 리소스 선점과 해제 모듈에서 코어 간 통신 인터페이스를 통해 제2 인터랙션 요청을 제1 운영체제에 전송하도록 트리거한다.
- [0656] 본 실시예에서는, 제2 시스템 제어 모듈이 제2 운영체제의 처리 리소스에 기반하여 리소스 인터랙션을 트리거하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0657] 예시적인 일 실시예에서는, 제2 시스템 제어 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하기도 하다.
- [0658] 제2 운영체제의 처리 리소스의 현재의 리소스 이용율이 제2 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0659] 제2 운영체제의 실행 중 서비스 및 제2 운영체제의 실행 대상인 서비스에 따라 제2 운영체제의 처리 리소스의 리소스 이용율이 제3 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0660] 본 실시예에서는, 제2 시스템 제어 모듈이 제2 운영체제에서 수행될 리소스 인터랙션을 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0661] 예시적인 일 실시예에서는, 임베디드 시스템은, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮지 않을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것으로 결정하고, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것으로 결정한다.
- [0662] 본 실시예에서는, 임베디드 시스템이 제1 운영체제에서 제2 운영체제가 처리 리소스를 점유하는 것을 허용하는지 여부를 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0663] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 코어 간 통신 인터페이스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송하는 것을 실현한다.
- [0664] 코어 간 통신 인터페이스를 통해 중단 번호가 제3 중단 번호인 제2 선점 요청을 제1 운영체제에 전송하되, 제2 선점 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0665] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 중단을 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0666] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하여 코어 간 통신 인터페이스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하는 것을 실현한다.
- [0667] 코어 간 통신 인터페이스를 통해 중단 번호가 제4 중단 번호인 리소스 해제를 허용하는 응답을 제2 운영체제에 전송하되, 리소스 해제를 허용하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것을 지시하기 위한 것이다.
- [0668] 코어 간 통신 인터페이스를 통해 중단 번호가 제5 중단 번호인 리소스 해제를 거부하는 응답을 제2 운영체제에 전송하되, 리소스 해제를 거부하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것을 지시하기 위한 것이다.
- [0669] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 중단을 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0670] 예시적인 일 실시예에서는, 리소스 적응형 스케줄링 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하기도 하다. 제1 운영체제에 의해 선점된 제2 운영체제의 처리 리소스를 제1 운영체제의 사용 가능한 리소스풀에

병합하고, 제2 운영체제에 의해 선점된 제1 운영체제의 처리 리소스 또는 제1 운영체제에서 제2 운영체제에 능동적으로 해제한 처리 리소스를 제2 운영체제의 사용 가능한 리소스풀에 병합한다.

- [0671] 본 실시예에서는, 리소스 적응형 스케줄링 모듈이 서로 다른 운영체제 사이에서 처리 리소스 스케줄링을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0672] 예시적인 일 실시예에서는, 임베디드 시스템은 시스템 시작 모듈을 더 포함한다.
- [0673] 시스템 시작 모듈은, 프로세서가 위치하는 칩의 전원이 켜진 후, 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하고, 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제의 실행을 부트하되, 제1 초기 처리 리소스는 프로세서의 처리 리소스에서 제1 운영체제와 대응되는 초기 처리 리소스이고, 제2 초기 처리 리소스는 프로세서의 처리 리소스에서 제2 운영체제와 대응되는 초기 처리 리소스이다.
- [0674] 본 실시예에서는, 시스템 시작 모듈이 부트 로드 프로그램을 통해 대응되는 초기 처리 리소스에서 각 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0675] 예시적인 일 실시예에서는, 제1 초기 처리 리소스는 프로세서의 프로세서 코어에서의 하나의 지정된 프로세서 코어이고, 제2 초기 처리 리소스는 프로세서의 프로세서 코어에서 지정된 프로세서 코어와의 다른 프로세서 코어이다.
- [0676] 예시적인 일 실시예에서는, 시스템 시작 모듈은 하기와 같은 단계를 실행하여 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하는 것을 실현한다.
- [0677] 2차 프로그램 로더를 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트한다.
- [0678] 본 실시예에서는, 시스템 시작 모듈이 2차 프로그램 로더를 통해 초기 처리 리소스에서 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0679] 예시적인 일 실시예에서는, 시스템 시작 모듈은, 기기의 메인보드에 고정된 시작 칩에서의 사전 설정된 프로그램을 통해 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행하되, 2차 프로그램 로더는 보안 시작 점검의 점검 결과가 정상인 후에 실행된다.
- [0680] 본 실시예에서는, 시스템 시작 모듈이 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0681] 예시적인 일 실시예에서는, 시스템 시작 모듈은 하기와 같은 단계를 실행하여 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제의 실행을 부트하는 것을 실현한다.
- [0682] 2차 프로그램 로더를 통해 제2 초기 처리 리소스를 웨이크업하고, 범용의 부트 로드 프로그램의 실행을 부트하여, 제2 운영체제의 실행을 부트한다.
- [0683] 본 실시예에서는, 시스템 시작 모듈이 2차 프로그램 로더를 통해 제2 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0684] 본 출원 실시예의 또 다른 측면에 따르면, 상기 임베디드 시스템의 실행 방법을 실시하기 위한 임베디드 시스템을 더 제공한다. 도 10은 본 출원의 실시예에 따른 선택적인 임베디드 시스템의 예시도이다. 상기 임베디드 시스템은,
- [0685] 칩 및 적어도 2개의 운영체제를 포함할 수 있다. 여기서,
- [0686] 칩은 프로세서(1002), 하드웨어 제어기(1004), 제1 버스(1006) 및 제2 버스(1008)를 포함한다. 여기서,
- [0687] 제1 버스(1006)의 대역폭은 제2 버스(1008)의 대역폭보다 높고, 또한 제1 버스(1006)는 멀티 마스터 멀티 슬레이브 모드로 구성되고, 제2 버스(1008)는 단일 마스터 멀티 슬레이브 모드로 구성된다.
- [0688] 적어도 2개의 운영체제는 프로세서(1002)에 기반하여 실행된다. 여기서, 프로세서의 처리 리소스는 적어도 2개의 운영체제에 동적 할당되고, 프로세서의 처리 리소스는 프로세서 코어를 포함한다.
- [0689] 적어도 2개의 운영체제는 제1 버스(1006)를 통해 통신을 수행하고;
- [0690] 적어도 2개의 운영체제는 제2 버스(1008)를 통해 하드웨어 제어기에 대한 제어를 실현한다.

- [0691] 여기서, 상기 칩은 상기 BMC 칩일 수 있으며; 상기 프로세서는 상기 멀티 코어 프로세서일 수 있고, 상기 하드웨어 제어기는 대응되는 외부 인터페이스에 연결되는 주변기기를 제어하며; 상기 제1 버스는 멀티 마스터 멀티 슬레이브 모드로 구성되고, 이는, 예를 들면, AHB(Advanced High Performance Bus, 선진적인 고성능 버스)와 같은 프로세서의 복수의 프로세서 코어 사이에서 통신하는데 사용되는 버스이며, 상기 제2 버스는 단일 마스터 멀티 슬레이브 모드로 구성되고, 이는, 예를 들면, APB(Advanced Peripheral Bus, 주변 버스)와 같은 프로세서와 하드웨어 제어기 사이를 제어하는데 사용되는 버스이며, 제1 버스의 대역폭은 제2 버스의 대역폭보다 높다.
- [0692] 여기서, 제1 버스가 멀티 마스터 멀티 슬레이브 모드로 구성되는 것은, 제1 버스가 복수의 마스터 기기(Master) 및 복수의 슬레이브 기기(Slave) 사이에서 데이터 통신을 수행하는 것을 가리킨다. 즉, 상기 제1 버스에는 복수의 마스터 기기와 복수의 슬레이브 기기가 연결되고, 마스터 기기와 마스터 기기 사이 및 마스터 기기와 슬레이브 기기 사이는 모두 상기 제1 버스를 사용하여 데이터 통신을 수행할 수 있다. 전반 제1 버스에서의 데이터 전송은 마스터 기기에 의해 발신되고, 슬레이브 기기에서는 응답을 담당한다. 제2 버스가 단일 마스터 멀티 슬레이브 모드로 구성되는 것은, 제2 버스가 하나의 마스터 기기와 복수의 슬레이브 기기 사이에서 데이터 통신을 수행하는 것을 가리킨다. 즉, 상기 제2 버스에는 하나의 마스터 기기와 복수의 슬레이브 기기가 연결되고, 마스터 기기와 슬레이브 기기 사이는 상기 제2 버스를 사용하여 데이터 통신을 수행할 수 있다. 데이터 요청은 마스터 기기에서 슬레이브 기기로 발신할 수밖에 없고, 슬레이브 기기는 요청을 수신한 후 대응되는 응답 데이터를 마스터 기기에 반환한다. 이 과정은 일대다의 액세스를 실현할 수 있다.
- [0693] 설명해야 할 것은, 운영체제 사이에서 멀티 마스터 멀티 슬레이브 모드의 버스를 사용하여 데이터 통신을 수행하여, 각 운영체제에서 수요에 따라 능동적으로 데이터 요청을 발신하는데 용이하도록 할 수 있고, 하드웨어 제어기는 주로 운영체제의 제어에 기반하여 대응되는 하드웨어를 제어할 수 있으므로, 운영체제와 하드웨어 제어기 사이는 단일 마스터 멀티 슬레이브 모드의 버스를 사용하여 통신을 수행하고, 데이터 요청은 모두 운영체제에서 하드웨어 제어기로 발신하여, 하드웨어 제어기의 제어 효율을 향상시킬 수 있다.
- [0694] 상기 AHB 버스는 AMBA(Advanced Microcontroller Bus Architecture, 선진적인 마이크로 제어기 버스 아키텍처) 2에서 이미 정의되었다. AHB 버스는 주로 시스템 고속 버스로서 사용되고, 고성능 및 저소모의 시스템 디자인에 적용되며, 이는 고성능 모듈 사이의 연결에 사용될 수 있다. 고성능 모듈은 SoC의 온칩 시스템 버스로서, CPU, DMA(Direct Memory Access, 직접 메모리 액세스) 및 DSP(Digital Signal Processing, 디지털 신호 프로세서) 등일 수 있다. AMBA 프로토콜에서, AHB는 주로 시스템 레벨의 고대역폭 고성능의 시스템 상호 접속을 지향하는 디자인으로서, 단일 클럭 에지 동작; 논 트리스테이트 실행방식(non-tristate implementation); 돌발적인 전송 지원; 분할 전송 지원; 멀티 master 멀티 slave의 상호 접속 모드 지원; 32비트~128비트 버스 폭으로 구성 가능; 바이트, 반바이트 및 문자 전송 지원; 의 특성을 포함한다. AHB 시스템은 마스터 모듈(즉, 마스터 기기), 슬레이브 모듈(즉, 슬레이브 기기) 및 기초 구조(Infrastructure)의 3부분을 포함한다. 전반 AHB 버스에서의 전송은 모두 마스터 모듈에서 발신되고, 슬레이브 모듈에서는 응답을 담당한다. 기초 구조는 아비터(arbiter), 마스터 모듈에서 슬레이브 모듈까지의 멀티 플렉서, 디코더(decoder), 가상 슬레이브 모듈(dummy Slave) 및 가상 마스터 모듈(dummy Master)을 포함한다.
- [0695] APB는, 예를 들면, UART, 1284 등과 같이 주로 낮은 대역폭의 주변기기 사이의 연결에 사용된다. 이의 버스 아키텍처는 AHB와 달리 복수의 마스터 모듈을 지원하고, APB에서 고유의 마스터 모듈이 바로 APB 브릿지이다. 이는 하기와 같은 특성을 포함한다.
- [0696] ① 고주파에서 작동될 수 있다.
- [0697] ② 프로토콜이 간단하고, 복잡한 타임 시퀀스가 없다.
- [0698] ③ 버스와 동기화 되고, 버스에서 모든 transaction(읽기/쓰기 동작)은 모두 클럭의 상승 엣지에 의존한다.
- [0699] ④ 단일 마스터 멀티 슬레이브이고, 일반적인 경우, APB는 AHB 버스 시스템에 걸려있고, AHB-APB Bridge를 통해 서비스를 AHB 버스 시스템 사이에서 변환하며, 이때 Bridge는 APB의 master이고, 다른 주변기기는 모두 slave이다.
- [0700] ⑤ 인터페이스가 간단하다. 즉 AXI 및 AHB에 비해, 인터페이스가 비교적 간단하다.
- [0701] ⑥ 전력소모가 낮다.
- [0702] ⑦ 예를 들면, 상기 하드웨어 제어기를 통해 제어 가능한 주변기기와 같은 다양한 주변기기를 연결할 수 있다.
- [0703] APB 버스의 경우, 데이터 요청은 Master에서 slave로 발신될 수밖에 없고, slave에서 요청을 수신한 후 대응되

는 응답 데이터를 Master에 반환한다. 이 과정은 일대다의 액세스를 실현할 수 있으며, 또한 액세스에는 AHB 버스에서의 중재 및 Decoder의 해석 동작이 포함되지 않는다. 여기서, AHB 버스는 높은 대역폭 특성을 갖고, 시스템에서 고성능 모듈(CPU, DMA 등) 간의 상호 접속에 사용된다. APB 버스는 대역폭이 낮고, 시스템에서 주변기기(UART 및 I2C 등) 사이의 연결에 사용된다. AHB 버스는 로직 회로 및 버스 프로토콜이 복잡하고, APB 버스는 인터페이스 회로 및 버스 프로토콜이 상대적으로 간단하다.

[0704] 임베디드 시스템은 적어도 2개의 운영체제를 포함할 수 있다. 적어도 2개의 운영체제는 프로세서에 기반하여 실행되고, 프로세서의 처리 리소스는 적어도 2개의 운영체제에 동적 할당되며, 프로세서의 처리 리소스는 프로세서 코어를 포함하고, 적어도 2개의 운영체제는 제1 버스를 통해 통신하며, 적어도 2개의 운영체제는 제2 버스를 통해 하드웨어 제어기에 대한 제어를 실현한다.

[0705] 선대적으로, 하드웨어 제어기는 한가지 또는 여러가지를 포함할 수 있으며, I2C, USB(Universal Serial Bus, 범용 직렬 버스), UART, ADC(Analog to Digital Converter, 아날로그 디지털 변환기), JTAG(Joint Test Action Group, 연합 테스트 동작 그룹), RTC(Real Time Clock, 실시간 클럭), GPIO(General Purpose Input/Output, 범용 입출력), WDT(Watch Dog Timer, 워치 도그 타이머), 가상 UART(Virtual UART), 슈퍼 I/O(Super I/O), SGPIO(Serial General Purpose Input/Output, 직렬 범용 입출력), PWM(Pulse Width Modulation, 펄스 폭 변조), FanTach(팬 속도 조절), Timer(시계), PECI(Platform Environment Control Interface, 플랫폼 환경식 제어 인터페이스), 메일박스(MailBox) 중의 적어도 하나의 칩 주변기기에 대응되는 제어기를 포함할 수 있지만, 이에 한정되지 않으며, 다른 유형의 제어기를 더 포함할 수 있다. 외부 인터페이스는 한가지 또는 여러가지를 포함할 수 있으며, 상기 어느 한 제어기에 대응되는 외부 인터페이스를 포함할 수 있지만, 이에 한정되지 않는다.

[0706] 예를 들면, BMC 칩의 일 예시는 도 11에 도시된 바와 같을 수 있으며, BMC 칩의 하드웨어는 SOC 서브 모듈 및 BMC 대역의 서브 모듈을 포함할 수 있지만 이에 한정되지 않는다. 여기서, SOC 서브 모듈은 주로 ARM 코어(ARM Core 1, ARM Core 2, ..., ARM Core X)를 포함하며, 이는 DDR(Double Data Rate, 더블 속도)4 제어기(메모리 제어기), MAC(Media Access Control Address, 미디어 액세스 제어 주소) 제어기(네트워크 제어기), SD(Secure Digital, 보안 디지털) Card/Emmc(Embedded Multi Media Card, 임베디드 멀티 미디어 카드) 제어기(자장 제어기), PCIe RC(Root Complex, 루트 컴플렉스) 제어기, SRAM(Static Random-Access Memory, 정적 랜덤 액세스 메모리) 및 SPI 제어기를 더 포함할 수 있지만, 이에 한정되지 않는다.

[0707] 상기 코어는 각 제어기와 제1 버스를 통해 상호 연결되어, 코어와 각 제어기 간의 인터랙션을 실현한다. 동시에 ARM 코어 간에는 제1 버스(예를 들면, AXI (Advanced eXtensible Interface, 선진적인 확장 가능한 인터페이스) 브릿지(Bridge)를 통해 연결될 수 있음)에 연결되고, 코어 간의 통신은 제1 버스를 통해 실현된다. 또한, SOC 서브 모듈에서 제1 버스와 제2 버스의 상호 연결 및 상호 연동(예를 들면, APB 브릿지(APB Bridge)의 변환으로 실현함)도 실현한다. 이렇게 SOC 서브 모듈이 제2 버스에서의 주변기기를 액세스하는데 물리적 채널을 제공한다.

[0708] DDR4 제어기는 DDR4 PHY(Physical Layer, 물리 계층) 인터페이스를 통해 다른 부품 또는 기기와 상호 연결되고, MAC 제어기는 RGMII(Reduced Gigabit Media Independent Interface, 기가비트 매체 독립 인터페이스)를 통해 다른 부품 또는 기기와 상호 연결되며, SD 카드/Emmc 제어기는 SD 인터페이스를 통해 다른 부품 또는 기기와 상호 연결되고, PCIe RC 제어기는 PCIe PHY 인터페이스를 통해 다른 부품 또는 기기와 상호 연결된다.

[0709] BMC 대역의 서브 모듈은 주로 PWM, GPIO, FanTech(팬 속도 조절), mailbox(메일 박스) 등 칩의 주변기기에 대응되는 제어기를 포함한다. 이러한 제어기를 통해 BMC에 대한 PECI 통신(예를 들면, GPIO 시뮬레이션 PECI를 사용함), 팬 조절 및 제어 등 대역의 관리 기능을 실현할 수 있다. 도 12에서 알 수 있다 싶이, 상기 BMC 대역의 서브 모듈은 제2 버스를 통해 SOC 서브 모듈과의 인터랙션을 실현하는 것을 포함할 수 있지만, 이에 한정되지 않는다.

[0710] BMC 칩은 제1 버스와 제2 버스를 통해 온칩 ARM 코어, 저장 유닛 및 제어기 하드웨어 리소스 간의 상호 연결을 실현한다. 프로세서 리소스의 동적 밸런싱 스케줄링은 주로 BMC 칩의 ARM 코어 리소스 스케줄링에 관한 것며, 코어 간 통신은 ARM 코어 사이에서 수행되는 통신을 가리킨다. Linux 시스템에서 RTOS 시스템 코어를 선점하는 것을 예로 들면, Linux 시스템에서 먼저 코어2 내지 코어N의 어느 한 코어에서 칩은 제1 버스를 통해 코어1로 코어 간 중단(중단 번호 9)을 발신한다. 이때 RTOS 시스템이 유휴 상태일 경우 선점을 허용하고, 코어1에서는 제1 버스를 통해 코어 간 중단(중단 번호 10)을 회신하며, 현재 코어1에 의해 매핑된 주변기기 제어기 리소스(예를 들면, PWM/PECI)를 해제하고, Linux 시스템에서는 코어 간 중단 10을 수신하며, 선점 흐름을 발신하고,

코어1을 Linux SMP 스케줄링에 추가하여, 동시에 PWM/PECI 주변기기의 제어 권한을 획득하며, 제2 버스를 통해 이를 제어할 수 있다.

- [0711] 일 측면에서, 적어도 2개의 운영체제는 제1 운영체제 및 제2 운영체제를 포함하며, 칩은 통신값을 제1 버스에 로드하고, 제1 버스는 통신값을 갖는 통신 신호를 제2 운영체제에 대응되는 레지스터에 발신하여, 제1 운영체제와 제2 운영체제 사이의 통신을 실현한다. 여기서, 통신값은 제1 운영체제와 제2 운영체제 사이의 통신 콘텐츠를 지시하기 위한 것이다.
- [0712] 다른 일 측면에서, 칩은 제어값을 제2 버스에 로드하고, 제2 버스는 제어값을 갖는 제어 신호를 하드웨어 제어기에 대응되는 레지스터에 발신하여, 운영체제에서 하드웨어 제어기에 대한 제어를 실현한다. 여기서, 제어값은 운영체제에서 하드웨어 제어기에 대한 제어 콘텐츠를 지시하기 위한 것이다.
- [0713] 운영체제에서는 각 하드웨어 제어기의 레지스터에 대한 액세스(예를 들면, 읽기 동작과 쓰기 동작을 실행함)를 통해 하드웨어 제어기를 제어하고, 운영체제에서 하드웨어 제어기의 레지스터를 액세스하는 방식은, 각 하드웨어 제어기의 레지스터 주소를 읽거나 쓰는 것일 수 있지만, 이에 한정되지 않는다. 이러한 레지스터의 주소는 칩 디자인 시 고유하고 또한 결정된 것일 수 있지만, 이에 한정되지 않는다. 예를 들면, 운영체제에서 특정의 주소(즉, 상기 통신 레지스터 또는 하드웨어 제어기에 대응되는 레지스터)에 특정의 값(즉, 상기 통신값 또는 제어값)을 쓰면 특정의 기능(예를 들면, 상기 운영체제 사이의 통신 기능 또는 운영체제에서 하드웨어 제어기에 대한 제어 기능)을 실현할 수 있다. 다시말하면, 서로 다른 기능은 서로 다른 제어값에 대응되고, 칩에서는 하드웨어 제어기의 기능과 제어값 사이의 대응관계를 유지보수한다. 예를 들면, 제어값 00은 어어컨을 1단 가속시키는 것을 나타내고, 제어값 01은 에어컨을 1단 감속시키는 것을 나타낸다.
- [0714] 각 운영체제 사이 및 운영체제와 하드웨어 제어기 사이는 버스를 통해 통신 및 제어 등등의 인터랙션을 수행할 수 있지만, 이에 한정되지 않는다. 상기 운영체제에서 각 하드웨어 제어기의 레지스터에 대한 읽기/쓰기 동작은 최종적으로 제1 버스(또는 제2 버스)에서 상기 하드웨어 제어기에 대한 제어 신호로 변환한다. 이부분의 변환 작동 및 제1 버스(또는 제2 버스)에서 하드웨어 제어기에 대한 제어 과정은 칩 내부 하드웨어에서 자동으로 실현할 수 있지만, 이에 한정되지 않는다. 그 실현 과정은 버스 스펙을 따른다. 여기서, 제1 버스(또는 제2 버스)의 동작 과정에서 일 측면으로는 제어와 버스 프로토콜이 관련되는 물리적 신호를 전송할 수 있으며, 다른 일 측면으로는 물리적 데이터 채널을 통해 유효한 데이터를 각 하드웨어 제어기에 전송할 수 있다.
- [0715] 상기 임베디드 시스템에서, 상기 임베디드 시스템은, 칩 및 적어도 2개의 운영체제를 포함한다. 여기서, 칩은 프로세서, 하드웨어 제어기, 제1 버스 및 제2 버스를 포함한다. 여기서, 제1 버스의 대역폭은 제2 버스의 대역폭보다 높고, 또한 제1 버스는 멀티 마스터 멀티 슬레이브 모드로 구성되고, 제2 버스는 단일 마스터 멀티 슬레이브 모드로 구성된다. 적어도 2개의 운영체제는 프로세서에 기반하여 실행된다. 여기서, 프로세서의 처리 리소스는 적어도 2개의 운영체제에 동적 할당되고, 프로세서의 처리 리소스는 프로세서 코어를 포함한다. 적어도 2개의 운영체제는 제1 버스를 통해 통신을 수행하고, 적어도 2개의 운영체제는 제2 버스를 통해 하드웨어 제어기에 대한 제어를 실현한다. 따라서, 관련 기술에 존재하는 멀티 코어 프로세서 다수의 처리 리소스가 유휴 상태인 것으로 인한 코어 리소스의 전반 이용율이 낮은 과제를 해결함으로써, 처리 리소스의 이용율을 향상시킨다.
- [0716] 예시적인 일 실시예에서는, 적어도 2개의 운영체제는 프로세서의 적어도 하나의 프로세서 코어에서 실행되는 제1 운영체제 및 적어도 하나의 프로세서 코어에서 실행되는 제2 운영체제를 포함한다. 제1 운영체제의 응답 속도는 제2 운영체제보다 빠르다. 여기서, 제2 운영체제는 서비스 관리 모듈, 리소스 동적 할당 모듈 및 리소스 적응형 스케줄링 모듈을 포함한다.
- [0717] 서비스 관리 모듈은, 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당한다. 여기서, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율 중의 적어도 하나에 따라 리소스 동적 할당을 수행한다.
- [0718] 리소스 동적 할당 모듈은, 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정한다. 여기서, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이다.
- [0719] 리소스 적응형 스케줄링 모듈은, 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.
- [0720] 본 실시예에서는, 적어도 2개의 운영체제는 제1 운영체제 및 제2 운영체제를 포함하며, 운영체제 각각은 적어도 하나의 프로세서 코어에서 실행된다. 제1 운영체제 및 제2 운영체제는 상기 실시예와 유사하며, 본 실시예에서

는 이에 대해 설명을 생략한다.

- [0721] 예시적인 일 실시예에서는, 제2 운영체제는 로드 밸런스 전략 모듈을 더 포함한다.
- [0722] 로드 밸런스 전략 모듈은, 규칙 구성 파일을 읽고, 리소스 동적 할당 규칙을 기록하기 위한 규칙 구조체를 생성한다.
- [0723] 본 실시예에서는, 로드 밸런스 전략 모듈이 규칙 구조체를 생성하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0724] 예시적인 일 실시예에서는, 로드 밸런스 전략 모듈은 또한, 제2 운영체제의 외부 인터페이스를 통해 규칙 업데이트 구성 파일을 획득한다. 여기서, 규칙 업데이트 구성 파일은 구성된 리소스 동적 할당 규칙을 업데이트하기 위한 것이고, 규칙 업데이트 구성 파일을 사용하여 규칙 구조체를 업데이트하여, 규칙 구조체에 기록된 리소스 동적 할당 규칙을 업데이트한다.
- [0725] 본 실시예에서는, 로드 밸런스 전략 모듈에 저장된 리소스 동적 할당 규칙은 제2 운영체제의 외부 인터페이스를 통해 원활하게 구성될 수 있고, 구성 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0726] 예시적인 일 실시예에서는, 서비스 관리 모듈은 하기와 같은 단계를 실행하여 리소스 동적 할당 규칙에서 서비스 응답 속도와 대응되는 할당 규칙에 따라 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0727] 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0728] 본 실시예에서는, 서비스 관리 모듈이 서비스 응답 속도에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0729] 예시적인 일 실시예에서는, 서비스 관리 모듈은 하기와 같은 단계를 실행하여 리소스 동적 할당 규칙에서 서비스 서비스 리소스 점유율과 대응되는 할당 규칙에 따라 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0730] 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0731] 본 실시예에서는, 서비스 관리 모듈이 서비스 리소스 점유율에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0732] 예시적인 일 실시예에서는, 서비스 관리 모듈은 또한 하기와 같은 단계 중의 적어도 한 단계를 실행하여 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0733] 한세트의 할당 대상인 서비스에서 제1 운영체제의 할당된 서비스와의 서비스 결합도가 제1 결합도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당한다.
- [0734] 한세트의 할당 대상인 서비스에서 제2 운영체제의 할당된 서비스와의 서비스 결합도가 제2 결합도 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0735] 본 실시예에서는, 서비스 관리 모듈이 서비스 결합도에 따라 할당 대상인 서비스를 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0736] 예시적인 일 실시예에서는, 서비스 관리 모듈은 또한 하기와 같은 단계를 실행하여 한세트의 할당 대상인 서비스를 대응되는 운영체제에 할당하는 것을 실현한다.
- [0737] 한세트의 할당 대상인 서비스에 민감한 정보가 포함되는 할당 대상인 서비스를 타깃 운영체제에 할당한다. 여기서, 타깃 운영체제는 제1 운영체제 및 제2 운영체제에서 사용 객체와 인터랙션하는 빈도가 낮은 운영체제이다.
- [0738] 본 실시예에서는, 서비스 관리 모듈이 민감한 정보를 포함하는 할당 대상인 서비스를 제1 운영체제에 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0739] 예시적인 일 실시예에서는, 리소스 동적 할당 모듈은 하기와 같은 단계를 실행하여 한세트의 할당 대상인 서비

스와 대응되는 리소스 할당 결과를 결정하는 것을 실현한다.

- [0740] 서비스 관리 모듈의 할당 결과에 따라, 제1 운영체제의 처리 리소스의 리소스 이용 상황 및 제2 운영체제의 처리 리소스의 리소스 이용 상황을 결합하여, 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성한다.
- [0741] 본 실시예에서는, 리소스 동적 할당 모듈에서 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0742] 예시적인 일 실시예에서는, 리소스 동적 할당 모듈은 프로세서 코어를 단위로 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.
- [0743] 예시적인 일 실시예에서는, 리소스 적응형 스케줄링 모듈은 하기와 같은 단계를 실행하여 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 것을 실현한다.
- [0744] 리소스 할당 결과에 따라 프로세서의 처리 리소스에서의 할당되지 않은 처리 리소스에는 대응되는 할당 대상인 서비스가 존재하는 것으로 결정할 경우, 할당되지 않은 처리 리소스를 할당되지 않은 처리 리소스와 대응되는 할당 대상인 서비스에 의해 할당되는 운영체제에 할당한다.
- [0745] 본 실시예에서는, 리소스 적응형 스케줄링 모듈에서 할당되지 않은 처리 리소스를 운영체제에 할당하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0746] 예시적인 일 실시예에서는, 임베디드 시스템은 리소스 선점과 해제 모듈을 더 포함한다.
- [0747] 리소스 선점과 해제 모듈은, 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0748] 본 실시예에서는, 리소스 선점과 해제 모듈이 서로 다른 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0749] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 제1 버스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하고, 제1 버스는 프로세서의 서로 다른 프로세서 코어 사이에서 통신을 수행하는 버스이다.
- [0750] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 제1 버스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행하는 것을 실현한다.
- [0751] 제1 버스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송하되, 제1 인터랙션 요청은 제2 운영체제와 리소스 인터랙션을 수행하는 것을 요청하기 위한 것이고, 리소스 인터랙션은 리소스 선점 및 리소스 해제 중의 하나를 포함한다.
- [0752] 제1 버스를 통해 제2 운영체제에서 제1 인터랙션 요청에 응답하여 반환하는 제1 인터랙션 응답을 획득하되, 제1 인터랙션 응답은 제1 운영체제에서 제1 인터랙션 응답에 따라 제2 운영체제와 리소스 인터랙션을 수행하는 것을 지시하기 위한 것이다.
- [0753] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0754] 예시적인 일 실시예에서는, 제1 운영체제는 제1 시스템 제어 모듈을 더 포함한다.
- [0755] 제1 시스템 제어 모듈은, 제1 운영체제의 처리 리소스의 리소스 이용율을 검출하되, 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 제1 운영체제에서 제2 운영체제와 리소스 인터랙션을 수행하는 것을 대기하는 것으로 결정할 경우, 리소스 선점과 해제 모듈에서 제1 버스를 통해 제1 인터랙션 요청을 제2 운영체제에 전송하는 것을 트리거한다.
- [0756] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제의 처리 리소스의 처리 이용율에 기반하여 리소스 인터랙션을 트리거하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0757] 예시적인 일 실시예에서는, 제1 시스템 제어 모듈은, 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 제1 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정하는 단계; 제1 운영체제의 처리 리소스가 유휴 상태이고, 제1 운영체제에서 실행 대상인 서비스가 존재하지 않

을 경우, 제1 운영체제에서 제2 운영체제로 처리 리소스를 해제하는 것을 대기하는 것으로 결정하는 단계; 중의 적어도 한 단계를 실행하기도 하다.

- [0758] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제에서 수행할 리소스 인터랙션 유형을 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0759] 예시적인 일 실시예에서는, 제1 시스템 제어 모듈은 또한, 제1 운영체제에 스케줄링할 서비스가 없고, 제1 운영체제에 할당 대상인 서비스가 없을 경우, 제1 운영체제가 휴면 상태에 진입하도록 제어한다.
- [0760] 본 실시예에서는, 제1 시스템 제어 모듈이 제1 운영체제에서 휴면 상태에 진입하도록 제어하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0761] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하여 제1 버스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송하는 것을 실현한다.
- [0762] 제1 버스를 통해 중단 번호가 제1 중단 번호인 제1 선점 요청을 제2 운영체제에 전송하되, 제1 선점 요청은 제2 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0763] 제1 버스를 통해 중단 번호가 제2 중단 번호인 리소스 해제 요청을 제2 운영체제에 전송하되, 리소스 해제 요청은 제2 운영체제에 제1 운영체제에 의해 점유된 처리 리소스를 해제하는 것을 요청하기 위한 것이다.
- [0764] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 중단을 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0765] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 제1 버스를 통해 제1 운영체제와 제2 운영체제 사이에서 처리 리소스의 서점과 해제를 수행하는 것을 실현한다.
- [0766] 제1 버스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송하되, 제2 인터랙션 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0767] 제1 버스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하되, 제2 인터랙션 응답은 제1 운영체제가 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는지 여부를 지시하기 위한 것이다.
- [0768] 본 실시예에서는, 리소스 선점과 해제 모듈이 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0769] 예시적인 일 실시예에서는, 제2 운영체제는 제2 시스템 제어 모듈을 더 포함한다.
- [0770] 제2 시스템 제어 모듈은, 제2 운영체제의 처리 리소스의 리소스 이용율을 검출하되, 제2 운영체제의 처리 리소스의 리소스 이용율에 따라, 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정할 경우, 리소스 선점과 해제 모듈에서 제1 버스를 통해 제2 인터랙션 요청을 제1 운영체제에 전송하도록 트리거한다.
- [0771] 본 실시예에서는, 제2 시스템 제어 모듈이 제2 운영체제의 처리 리소스에 기반하여 리소스 인터랙션을 트리거하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0772] 예시적인 일 실시예에서는, 제2 시스템 제어 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하기도 하다.
- [0773] 제2 운영체제의 처리 리소스의 현재의 리소스 이용율이 제2 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0774] 제2 운영체제의 실행 중 서비스 및 제2 운영체제의 실행 대상인 서비스에 따라 제2 운영체제의 처리 리소스의 리소스 이용율이 제3 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0775] 본 실시예에서는, 제2 시스템 제어 모듈이 제2 운영체제에서 수행될 리소스 인터랙션을 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0776] 예시적인 일 실시예에서는, 제1 운영체제는 또한, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮지 않을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것으로 결정하고, 제1 운영체제의 실행 중인 서비스의 서비스

스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것으로 결정한다.

- [0777] 본 실시예에서는, 제1 운영체제에서 제2 운영체제가 처리 리소스를 점유하는 것을 허용하는지 여부를 결정하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0778] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계를 실행하여 제1 버스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송하는 것을 실현한다.
- [0779] 제1 버스를 통해 중단 번호가 제3 중단 번호인 제2 선점 요청을 제1 운영체제에 전송하되, 제2 선점 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0780] 본 실시예에서는, 리소스 선점과 해제 모듈이 제1 버스를 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0781] 예시적인 일 실시예에서는, 리소스 선점과 해제 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하여 제1 버스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득하는 것을 실현한다.
- [0782] 제1 버스를 통해 중단 번호가 제4 중단 번호인 리소스 해제를 허용하는 응답을 제2 운영체제에 전송하되, 리소스 해제를 허용하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것을 지시하기 위한 것이다.
- [0783] 제1 버스를 통해 중단 번호가 제5 중단 번호인 리소스 해제를 거부하는 응답을 제2 운영체제에 전송하되, 리소스 해제를 거부하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것을 지시하기 위한 것이다.
- [0784] 본 실시예에서는, 리소스 선점과 해제 모듈이 제1 버스를 통해 리소스 인터랙션을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0785] 예시적인 일 실시예에서는, 리소스 적응형 스케줄링 모듈은 하기와 같은 단계 중의 적어도 한 단계를 실행하기도 하다. 제1 운영체제에 의해 선점된 제2 운영체제의 처리 리소스를 제1 운영체제의 사용 가능한 리소스풀에 병합하고, 제2 운영체제에 의해 선점된 제1 운영체제의 처리 리소스 또는 제1 운영체제에서 제2 운영체제에 능동적으로 해제한 처리 리소스를 제2 운영체제의 사용 가능한 리소스풀에 병합한다.
- [0786] 본 실시예에서는, 리소스 적응형 스케줄링 모듈이 서로 다른 운영체제 사이에서 처리 리소스 스케줄링을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0787] 예시적인 일 실시예에서는, 프로세서는, 칩의 전원이 켜진 후, 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하고, 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제의 실행을 부트하되, 제1 초기 처리 리소스는 프로세서의 처리 리소스에서 제1 운영체제와 대응되는 초기 처리 리소스이고, 제2 초기 처리 리소스는 프로세서의 처리 리소스에서 제2 운영체제와 대응되는 초기 처리 리소스이다.
- [0788] 본 실시예에서는, 프로세서가 부트 로드 프로그램을 통해 대응되는 초기 처리 리소스에서 각 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0789] 예시적인 일 실시예에서는, 제1 초기 처리 리소스는 프로세서의 프로세서 코어에서의 하나의 지정된 프로세서 코어이고, 제2 초기 처리 리소스는 프로세서의 프로세서 코어에서 지정된 프로세서 코어와의 다른 프로세서 코어이다.
- [0790] 예시적인 일 실시예에서는, 프로세서는 하기와 같은 단계를 실행하여 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하는 것을 실현한다.
- [0791] 2차 프로그램 로더를 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트한다.
- [0792] 본 실시예에서는, 프로세서가 2차 프로그램 로더를 통해 초기 처리 리소스에서 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0793] 예시적인 일 실시예에서는, 프로세서는, 기기의 메인보드에 고정된 시작 칩에서의 사전 설정된 프로그램을 통해 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행하되, 2차 프로그램 로더는 보안 시작 점검의 점검 결

과가 정상인 후에 실행된다.

- [0794] 본 실시예에서는, 프로세서가 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0795] 예시적인 일 실시예에서는, 프로세서는 하기와 같은 단계를 실행하여 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제의 실행을 부트하는 것을 실현한다.
- [0796] 2차 프로그램 로더를 통해 제2 초기 처리 리소스를 웨이크업하고, 범용의 부트 로드 프로그램의 실행을 부트하여, 제2 운영체제의 실행을 부트한다.
- [0797] 본 실시예에서는, 프로세서가 2차 프로그램 로더를 통해 제2 운영체제의 실행을 부트하는 방식은 상기 실시예와 유사하며, 여기에서 설명을 생략한다.
- [0798] 본 출원 실시예의 또 다른 측면에 따르면, 상기 임베디드 시스템의 실행 방법을 실시하기 위한 임베디드 시스템의 실행 장치를 더 제공한다. 도 12는 본 출원의 실시예에 따른 선택적인 임베디드 시스템의 실행 장치의 구조 블록도이다. 도 12에 도시된 바와 같이, 상기 장치는 제1 할당 유닛(1202), 제1 결정 유닛(1206) 및 제2 할당 유닛(1206)을 포함할 수 있다.
- [0799] 제1 할당 유닛(1202)은, 리소스 동적 할당 규칙에 따라, 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당한다. 여기서, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 제1 운영체제와 제2 운영체제는 프로세서에서 실행되고, 제1 운영체제의 응답 속도는 제2 운영체제보다 빠르다.
- [0800] 제1 결정 유닛(1204)은, 제1 분할 유닛(1202)과 서로 연결되고, 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정한다. 여기서, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함한다.
- [0801] 제2 할당 유닛(1206)은, 제1 결정 유닛(1204)와 서로 연결되고, 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당한다.
- [0802] 설명해야 할 것은, 상기 실시예에서의 제1 할당 유닛(1202)은 상기 단계 S202를 실행하는데 사용될 수 있고, 상기 실시예에서 제1 결정 유닛(1204)은 상기 단계 S204를 실행하는데 사용될 수 있으며, 상기 실시예에서 제2 할당 유닛(1206)은 상기 단계 S206을 실행하는데 사용될 수 있다.
- [0803] 상기 모듈에서, 리소스 동적 할당 규칙에 따라 한세트의 할당 대상인 서비스를 임베디드 시스템에서 대응되는 운영체제에 할당하되, 리소스 동적 할당 규칙은 서비스 응답 속도, 서비스 리소스 점유율, 서비스 결합도 및 서비스 중요성 중의 적어도 하나에 따라 리소스 동적 할당을 수행하는 것을 포함하고, 임베디드 시스템은 제1 운영체제 및 제2 운영체제를 포함하며, 제1 운영체제와 제2 운영체제는 프로세서에서 실행되고, 제1 운영체제의 응답 속도는 제2 운영체제보다 빠르며; 한세트의 할당 대상인 서비스와 대응되는 리소스 할당 결과를 결정하되, 리소스 할당 결과는 프로세서의 처리 리소스에서 한세트의 할당 대상인 서비스 중의 할당 대상인 서비스 각각에 대응되는 처리 리소스를 지시하기 위한 것이고, 프로세서의 처리 리소스는 프로세서 코어를 포함하는 것이며; 할당 대상인 서비스 각각과 대응되는 운영체제 및 리소스 할당 결과에 따라, 프로세서의 처리 리소스를 제1 운영체제 및 제2 운영체제에 할당하는 것; 을 통해, 관련 기술에 존재하는 멀티 코어 프로세서 다수의 처리 리소스가 유휴 상태인 것으로 인한 코어 리소스의 전반 이용율이 낮은 과제를 해결함으로써, 처리 리소스의 이용율을 향상시킨다.
- [0804] 예시적인 일 실시예에서는, 상기 장치는 생성 유닛을 더 포함한다.
- [0805] 생성 유닛은, 규칙 구성 파일을 읽고, 리소스 동적 할당 규칙을 기록하기 위한 규칙 구조체를 생성한다.
- [0806] 예시적인 일 실시예에서는, 상기 장치는 획득 유닛 및 업데이트 유닛을 더 포함한다.
- [0807] 획득 유닛은 또한, 제2 운영체제의 외부 인터페이스를 통해 규칙 업데이트 구성 파일을 획득한다. 여기서, 규칙 업데이트 구성 파일은 구성된 리소스 동적 할당 규칙을 업데이트하기 위한 것이다.
- [0808] 업데이트 유닛은, 규칙 업데이트 구성 파일을 사용하여 규칙 구조체를 업데이트하여, 규칙 구조체에 기록된 리

소스 동적 할당 규칙을 업데이트한다.

- [0809] 예시적인 일 실시예에서는, 제1 할당 유닛은 제1 할당 모듈, 제2 할당 모듈 및 제3 할당 모듈 중의 적어도 하나를 포함한다.
- [0810] 제1 할당 모듈은, 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 응답 속도 요구사항이 설정된 응답 속도 임계값보다 작은 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0811] 제2 할당 모듈은, 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값보다 작은 할당 대상인 서비스를 제1 운영체제에 할당하고, 한세트의 할당 대상인 서비스에서 서비스 리소스 점유율이 제1 점유율 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0812] 제3 할당 모듈은, 한세트의 할당 대상인 서비스에서 민감한 정보를 포함하는 할당 대상인 서비스를 타깃 운영체제에 할당한다. 여기서, 타깃 운영체제는 제1 운영체제 및 제2 운영체제에서, 사용 객체와 인터랙션 빈도가 낮은 운영체제이다.
- [0813] 예시적인 일 실시예에서는, 제1 할당 유닛은 제4 할당 모듈 및 제5 할당 모듈 중 적어도 하나를 포함한다.
- [0814] 제4 할당 모듈은, 한세트의 할당 대상인 서비스에서 제1 운영체제의 할당된 서비스와의 서비스 결합도가 제1 결합도 임계값 이상인 할당 대상인 서비스를 제1 운영체제에 할당한다.
- [0815] 제5 할당 모듈은, 한세트의 할당 대상인 서비스에서 제2 운영체제의 할당된 서비스와의 서비스 결합도가 제2 결합도 임계값 이상인 할당 대상인 서비스를 제2 운영체제에 할당한다.
- [0816] 예시적인 일 실시예에서는, 제1 결정 유닛은 생성 모듈을 포함한다.
- [0817] 생성 모듈은, 한세트의 할당 대상인 서비스 할당 결과에 따라, 제1 운영체제의 처리 리소스의 리소스 이용 상황 및 제2 운영체제의 처리 리소스의 리소스 이용 상황을 결합하여, 한세트의 할당 대상인 서비스와 프로세서의 처리 리소스의 리소스 매핑 테이블을 생성한다.
- [0818] 예시적인 일 실시예에서는, 프로세서의 처리 리소스는 프로세서 코어를 단위로 제1 운영체제 및 제2 운영체제에 할당된다.
- [0819] 예시적인 일 실시예에서는, 제2 할당 유닛은 제6 할당 모듈을 포함한다.
- [0820] 제6 할당 모듈은, 리소스 할당 결과에 따라 프로세서의 처리 리소스에서의 할당되지 않은 처리 리소스에는 대응되는 할당 대상인 서비스가 존재하는 것으로 결정할 경우, 할당되지 않은 처리 리소스를 할당되지 않은 처리 리소스와 대응되는 할당 대상인 서비스에 의해 할당되는 운영체제에 할당한다.
- [0821] 예시적인 일 실시예에서는, 상기 장치는 제1 실행 유닛을 더 포함한다.
- [0822] 제1 실행 유닛은, 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0823] 예시적인 일 실시예에서는, 제1 실행 유닛은 실행 모듈을 포함한다.
- [0824] 실행 모듈은, 코어 간 통신 인터페이스를 통해 제1 운영체제 및 제2 운영체제 사이에서 처리 리소스의 선점과 해제를 수행한다.
- [0825] 예시적인 일 실시예에서는, 실행 모듈은 제1 전송 서브 모듈 및 제1 획득 서브 모듈을 포함한다.
- [0826] 제1 전송 서브 모듈은, 코어 간 통신 인터페이스를 통해 제1 운영체제의 제1 인터랙션 요청을 제2 운영체제에 전송한다. 여기서, 제1 인터랙션 요청은 제2 운영체제와 리소스 인터랙션을 수행하는 것을 요청하기 위한 것이고, 리소스 인터랙션은 리소스 선점 및 리소스 해제 중의 하나를 포함한다.
- [0827] 제1 획득 서브 모듈은, 코어 간 통신 인터페이스를 통해 제2 운영체제에서 제1 인터랙션 요청에 응답하여 반환하는 제1 인터랙션 응답을 획득한다. 여기서, 제1 인터랙션 응답은 제1 운영체제에서 제1 인터랙션 응답에 따라 제2 운영체제와 리소스 인터랙션을 수행하는 것을 지시하기 위한 것이다.
- [0828] 예시적인 일 실시예에서는, 상기 장치는 제1 검출 유닛, 제2 결정 유닛을 더 포함한다.
- [0829] 제1 검출 유닛은, 제1 운영체제의 처리 리소스의 리소스 이용율을 검출한다.
- [0830] 제2 결정 유닛은, 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 제1 운영체제에서 제2 운영체제와 리소스

스 인터랙션을 수행하는 것을 대기하는 것으로 결정할 경우, 코어 간 통신 인터페이스를 통해 제1 인터랙션 요청을 제2 운영체제에 전송하는 것을 트리거한다.

- [0831] 예시적인 일 실시예에서는, 상기 장치는 제3 결정 유닛 및 제4 결정 유닛 중의 적어도 하나를 더 포함한다.
- [0832] 제3 결정 유닛은, 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 제1 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0833] 제4 결정 유닛은, 제1 운영체제의 처리 리소스가 유휴 상태이고, 제1 운영체제에서 실행 대상인 서비스가 존재하지 않을 경우, 제1 운영체제에서 제2 운영체제로 처리 리소스를 해제하는 것을 대기하는 것으로 결정한다.
- [0834] 예시적인 일 실시예에서는, 상기 장치는 제어 유닛을 더 포함한다.
- [0835] 제어 유닛은, 제1 운영체제에 스케줄링할 서비스가 없고, 제1 운영체제에 할당 대상인 서비스가 없을 경우, 제1 운영체제가 휴면 상태에 진입하도록 제어한다.
- [0836] 예시적인 일 실시예에서는, 제1 전송 서버 모듈은 제1 전송 서버 유닛 및 제2 전송 서버 유닛을 포함한다.
- [0837] 제1 전송 서버 유닛은, 코어 간 통신 인터페이스를 통해 중단 번호가 제1 중단 번호인 제1 선점 요청을 제2 운영체제에 전송한다. 여기서, 제1 선점 요청은 제2 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0838] 제2 전송 서버 유닛은, 코어 간 통신 인터페이스를 통해 중단 번호가 제2 중단 번호인 리소스 해제 요청을 제2 운영체제에 전송한다. 여기서, 리소스 해제 요청은 제2 운영체제에 제1 운영체제에 의해 점유된 처리 리소스를 해제하는 것을 요청하기 위한 것이다.
- [0839] 예시적인 일 실시예에서는, 실행 모듈은 제2 전송 서버 모듈 및 제2 획득 서버 모듈을 포함한다.
- [0840] 제2 전송 서버 모듈은, 코어간 통신 인터페이스를 통해 제2 운영체제의 제2 인터랙션 요청을 제1 운영체제에 전송한다. 여기서, 제2 인터랙션 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0841] 제2 획득 서버 모듈은, 코어간 통신 인터페이스를 통해 제1 운영체제에서 제2 인터랙션 요청에 응답하여 반환하는 제2 인터랙션 응답을 획득한다. 여기서, 제2 인터랙션 응답은 제1 운영체제가 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는지 여부를 지시하기 위한 것이다.
- [0842] 예시적인 일 실시예에서는, 상기 장치는 제2 검출 유닛 및 제5 결정 유닛을 더 포함한다.
- [0843] 제2 검출 유닛은, 제2 운영체제의 처리 리소스의 리소스 이용율을 검출한다.
- [0844] 제5 결정 유닛은, 제2 운영체제의 처리 리소스의 리소스 이용율에 따라, 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정할 경우, 리소스 선점과 해제 모듈에서 코어간 통신 인터페이스를 통해 제2 인터랙션 요청을 제1 운영체제에 전송하도록 트리거한다.
- [0845] 예시적인 일 실시예에서는, 상기 장치는 제6 결정 유닛 및 제7 결정 유닛 중의 적어도 하나를 더 포함한다.
- [0846] 제6 결정 유닛은, 제2 운영체제의 처리 리소스의 현재의 리소스 이용율이 제2 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0847] 제7 결정 유닛은, 제2 운영체제의 실행 중 서비스 및 제2 운영체제의 실행 대상인 서비스에 따라 제2 운영체제의 처리 리소스의 리소스 이용율이 제3 이용율 임계값 이상인 경우, 제2 운영체제에서 제2 운영체제의 처리 리소스를 선점하는 것을 대기하는 것으로 결정한다.
- [0848] 예시적인 일 실시예에서는, 상기 장치는 제8 결정 유닛 및 제9 결정 유닛을 더 포함한다.
- [0849] 제8 결정 유닛은, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮지 않을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것으로 결정한다.
- [0850] 제9 결정 유닛은, 제1 운영체제의 실행 중인 서비스의 서비스 우선순위가 제2 운영체제의 실행 대상인 서비스의 서비스 우선순위보다 낮을 경우, 제1 운영체제는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것으로 결정한다.
- [0851] 예시적인 일 실시예에서는, 제2 전송 서버 모듈은 제3 전송 서버 유닛을 포함한다.

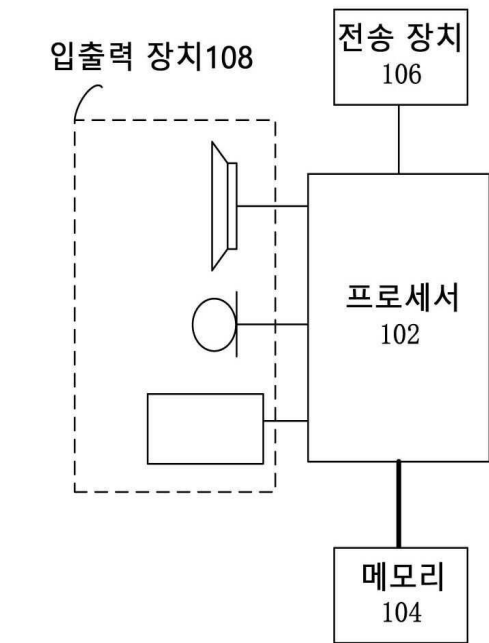
- [0852] 제3 전송 서브 유닛은, 코어 간 통신 인터페이스를 통해 중단 번호가 제3 중단 번호인 제2 선점 요청을 제1 운영체제에 전송한다. 여기서, 제2 선점 요청은 제1 운영체제의 처리 리소스를 선점하는 것을 요청하기 위한 것이다.
- [0853] 예시적인 일 실시예에서는, 제2 획득 서브 모듈은 제4 전송 서브 유닛 및 제5 전송 서브 유닛을 포함한다.
- [0854] 제4 전송 서브 유닛은, 코어 간 통신 인터페이스를 통해 중단 번호가 제4 중단 번호인 리소스 해제를 허용하는 응답을 제2 운영체제에 전송한다. 여기서, 리소스 해제를 허용하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 허용하는 것을 지시하기 위한 것이다.
- [0855] 제5 전송 서브 유닛은, 코어 간 통신 인터페이스를 통해 중단 번호가 제5 중단 번호인 리소스 해제를 거부하는 응답을 제2 운영체제에 전송하되, 리소스 해제를 거부하는 응답은 제1 운영체제에서는 제2 운영체제에서 제1 운영체제의 처리 리소스를 선점하는 것을 거부하는 것을 지시하기 위한 것이다.
- [0856] 예시적인 일 실시예에서는, 상기 장치는 제2 실행 유닛 및 제3 실행 유닛 중의 적어도 하나를 더 포함한다.
- [0857] 제2 실행 유닛은, 제1 운영체제에 의해 선점된 제2 운영체제의 처리 리소스를 제1 운영체제의 사용 가능한 리소스풀에 병합한다.
- [0858] 제3 실행 유닛은, 제2 운영체제에 의해 선점된 제1 운영체제의 처리 리소스 또는 제1 운영체제에서 제2 운영체제에 능동적으로 해제한 처리 리소스를 제2 운영체제의 사용 가능한 리소스풀에 병합한다.
- [0859] 예시적인 일 실시예에서는, 상기 장치는 부트 유닛을 더 포함한다.
- [0860] 부트 유닛은, 프로세서가 위치하는 칩의 전원이 켜진 후, 부트 로드 프로그램을 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트하고, 부트 로드 프로그램을 통해 제2 초기 처리 리소스에서 제2 운영체제의 실행을 부트한다. 여기서, 제1 초기 처리 리소스는 프로세서의 처리 리소스에서 제1 운영체제와 대응되는 초기 처리 리소스이고, 제2 초기 처리 리소스는 프로세서의 처리 리소스에서 제2 운영체제와 대응되는 초기 처리 리소스이다.
- [0861] 예시적인 일 실시예에서는, 제1 초기 처리 리소스는 프로세서의 프로세서 코어에서의 하나의 지정된 프로세서 코어이고, 제2 초기 처리 리소스는 프로세서의 프로세서 코어에서 지정된 프로세서 코어와의 다른 프로세서 코어이다.
- [0862] 예시적인 일 실시예에서는, 부트 유닛은 부트 모듈을 포함한다.
- [0863] 부트 모듈은, 2차 프로그램 로더를 통해 제1 초기 처리 리소스에서 제1 운영체제의 실행을 부트한다.
- [0864] 예시적인 일 실시예에서는, 상기 장치는 점검 유닛을 더 포함한다.
- [0865] 점검 유닛은, 기기의 메인보드에 고정된 시작 칩에서의 사전 설정된 프로그램을 통해 2차 프로그램 로더의 코드에 대해 보안 시작 점검을 수행한다. 여기서, 2차 프로그램 로더는 보안 시작 점검의 점검 결과가 정상인 후에 실행된다.
- [0866] 예시적인 일 실시예에서는, 부트 유닛은 실행 모듈을 더 포함한다.
- [0867] 실행 모듈은, 2차 프로그램 로더를 통해 제2 초기 처리 리소스를 웨이크업하고, 범용의 부트 로드 프로그램의 실행을 부트하여, 제2 운영체제의 실행을 부트한다.
- [0868] 설명해야 할 것은, 상기 각 모듈은 소프트웨어 또는 하드웨어를 통해 실현할 수 있는 것이며, 후자의 경우, 상기 모듈이 모두 동일한 프로세서에 위치하거나, 또는 상기 각 모듈이 임의의 조합의 형식으로 서로 다른 프로세서에 각각 위치하는 형태를 통해 실현할 수 있으며, 이에 한정되지 않는다.
- [0869] 본 출원의 실시예는 칩을 더 제공한다. 여기서, 상기 칩은 프로그래밍 가능한 로직 회로 및 실행 가능한 명령어 중의 적어도 하나를 포함한다. 상기 칩은 전자 기기에서 실행되며, 상술한 어느 한 항의 방법 실시예에서의 단계를 실현한다.
- [0870] 본 출원의 실시예는 BMC 칩을 더 제공한다. 여기서, 상기 BMC 칩은 저장 유닛 및 저장 유닛과 연결되는 처리 유닛을 포함할 수 있다. 저장 유닛은 프로그램을 저장하고, 처리 유닛은 상기 프로그램을 실행하여, 상술한 어느 한 항의 방법 실시예에서의 단계를 실행한다.
- [0871] 본 출원의 실시예는 메인보드를 더 제공한다. 여기서, 상기 메인보드는 적어도 하나의 프로세서 및 적어도 하나

의 메모리를 포함한다. 상기 적어도 하나의 메모리는, 적어도 하나의 프로그램을 저장한다. 적어도 하나의 프로그램이 적어도 하나의 프로세서에 의해 실행될 시, 적어도 하나의 프로세서에서 상술한 어느 한 항의 방법 실시예에서의 단계를 실현하도록 한다.

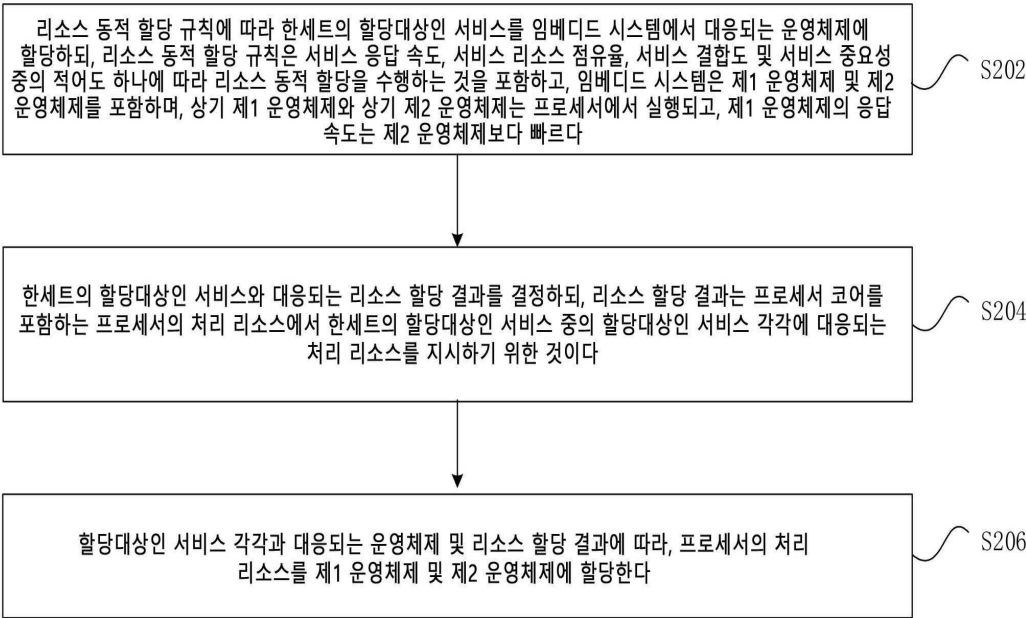
- [0872] 본 출원의 실시예는 서버를 더 제공한다. 상기 서버는, 프로세서, 통신 인터페이스, 메모리 및 통신 버스를 포함한다. 여기서, 프로세서, 통신 인터페이스 및 메모리는 통신 버스를 통해 상호 간의 통신을 완성한다. 메모리는 컴퓨터 프로그램을 저장하고, 프로세서는 메모리에 저장된 프로그램이 실행될 시, 상기 어느 한 항의 방법 실시예에서의 단계를 실현하여, 동일한 기술적 효과에 도달한다.
- [0873] 상기 서버의 통신 버스는 PCI(Peripheral Component Interconnect, 주변기기 부품 상호 연결 기준) 버스 또는 EISA(Extended Industry Standard Architecture, 확장 산업 기준 구조) 버스 등일 수 있다. 상기 통신 버스는 주소 버스, 데이터 버스, 제어 버스 등으로 나뉘 수 있다. 통신 인터페이스는 상기 서버와 다른 기기 사이의 통신을 위한 것이다.
- [0874] 메모리는 RAM을 포함할 수 있고, NVM(Non-Volatile Memory, 비휘발성 메모리), 예를 들면, 적어도 하나의 디스크 메모리를 포함할 수도 있다. 선택적으로, 메모리는 적어도 하나의 상기 프로세서와 멀리 떨어진 저장 장치일 수도 있다. 상기 프로세서는 CPU, NP(Network Processor, 네트워크 프로세서) 등을 포함하는 범용 프로세서일 수 있으며, DSP, ASIC, FPGA 또는 다른 프로그래밍 가능한 로직 디바이스, 분할 게이트 또는 트랜지스터 로직 디바이스, 분할 하드웨어 컴포넌트일 수도 있다.
- [0875] 본 출원의 실시예는 컴퓨터 판독가능 저장 매체를 더 제공한다. 상기 컴퓨터 판독가능 저장 매체에는 컴퓨터 프로그램이 저장되며, 상기 컴퓨터 프로그램은 실행 시 상술한 어느 한 항의 방법 실시예에서의 단계를 수행하도록 구성된다.
- [0876] 예시적인 일 실시예에서는, 상기 컴퓨터 판독가능 저장 매체는 USB 메모리, ROM, RAM, 이동식 하드디스크, 자기 디스크 또는 광 디스크 등 다양한 컴퓨터 프로그램을 저장할 수 있는 매체를 포함할 수 있지만, 이에 제한되지 않는다.
- [0877] 본 출원의 실시예는 전자 기기를 더 제공한다. 상기 전자 기기는 메모리 및 프로세서를 포함하며, 상기 메모리에는 컴퓨터 프로그램이 저장되고, 상기 프로세서는 컴퓨터 프로그램을 실행하여 상술한 어느 한 항의 방법 실시예에서의 단계를 실행하도록 구성된다.
- [0878] 예시적인 일 실시예에서는, 상기 전자 기기는 상기 프로세서와 연결되는 전송 기기 및 상기 프로세서와 연결되는 입출력 기기를 더 포함할 수 있다.
- [0879] 본 실시예에서의 구체적인 예시는 상기 실시예 및 예시적인 실시형태에서 설명된 예시를 참조할 수 있으며, 본 실시예는 여기에서 설명을 생략한다.
- [0880] 자명한 것은, 상기 본 출원의 각 모듈 또는 각 단계는 범용의 컴퓨터 장치로 실현할 수 있으며, 이들은 단일 컴퓨터 장치에 집중되거나, 또는 복수의 컴퓨터 장치로 구성된 네트워크에 분산될 수 있고, 이들은 컴퓨터 장치에서 실행될 수 있는 프로그램 코드로 실현할 수 있으므로, 이들을 저장 장치에 저장하여 컴퓨터 장치에서 실행할 수 있다. 또한 어떤 경우에는, 도시되거나 설명된 단계를 여기와 다른 순서로 실행하거나, 또는 이들을 각 집적회로 모듈로 각각 제조하거나, 또는 이들 중의 복수의 모듈 또는 단계를 단일 집적회로 모듈로 제조하여 실현할 수 있음을 당업자는 알아야 한다. 이와 같이, 본 출원은 그 어떤 특정된 하드웨어 및 소프트웨어의 결합에 한정되지 않는다.
- [0881] 이상은 본 출원의 바람직한 실시예일 뿐, 본 출원을 한정하기 위한 것은 아니며, 본 출원은 당업자에 의해 다양하게 변경 및 변화될 수 있다. 본 출원의 원칙 내에서 수행되는 임의의 수정, 균등교체, 개선 등은 모두 본 출원의 보호범위 내에 포함되어야 한다.

도면

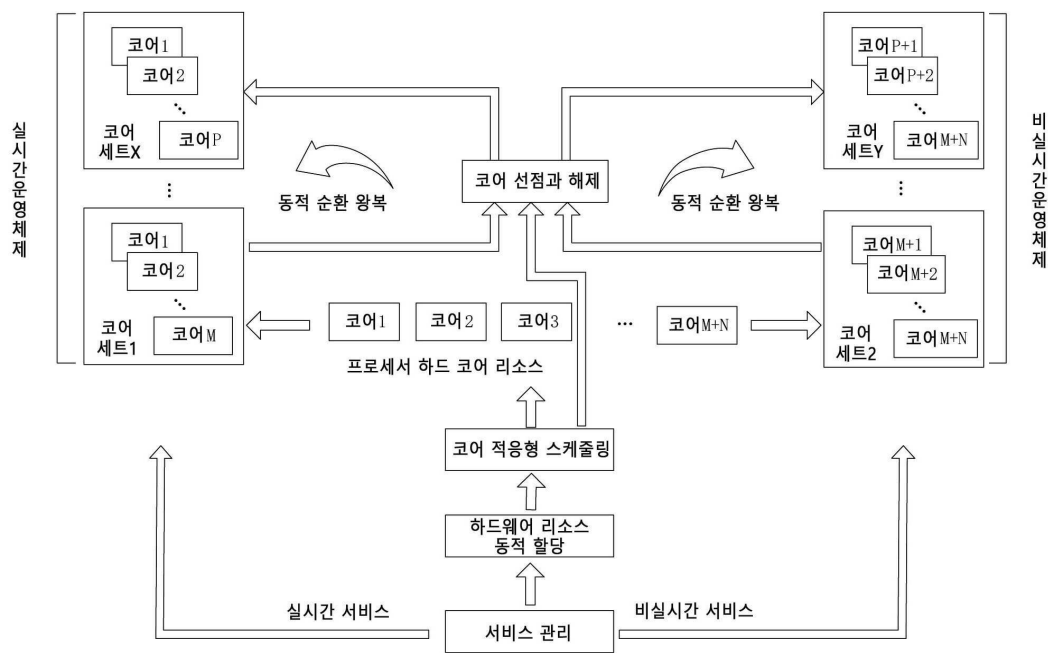
도면1



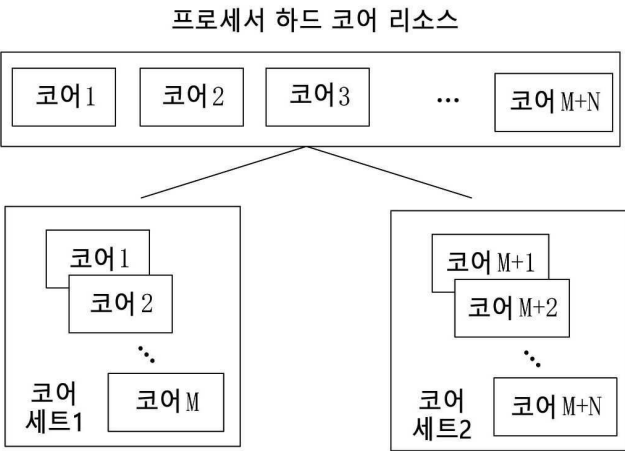
도면2



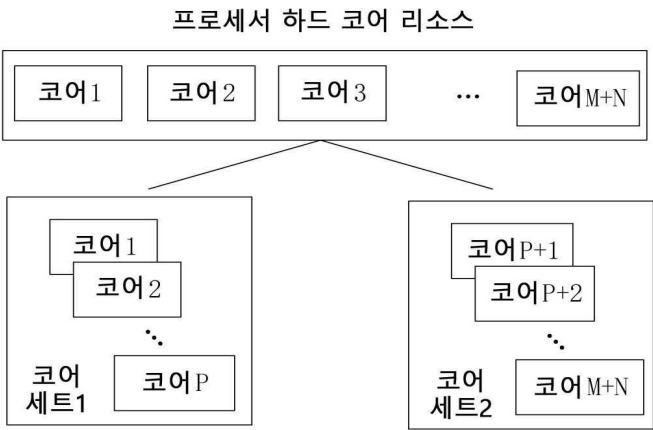
도면3



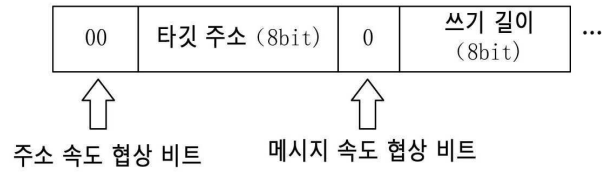
도면4



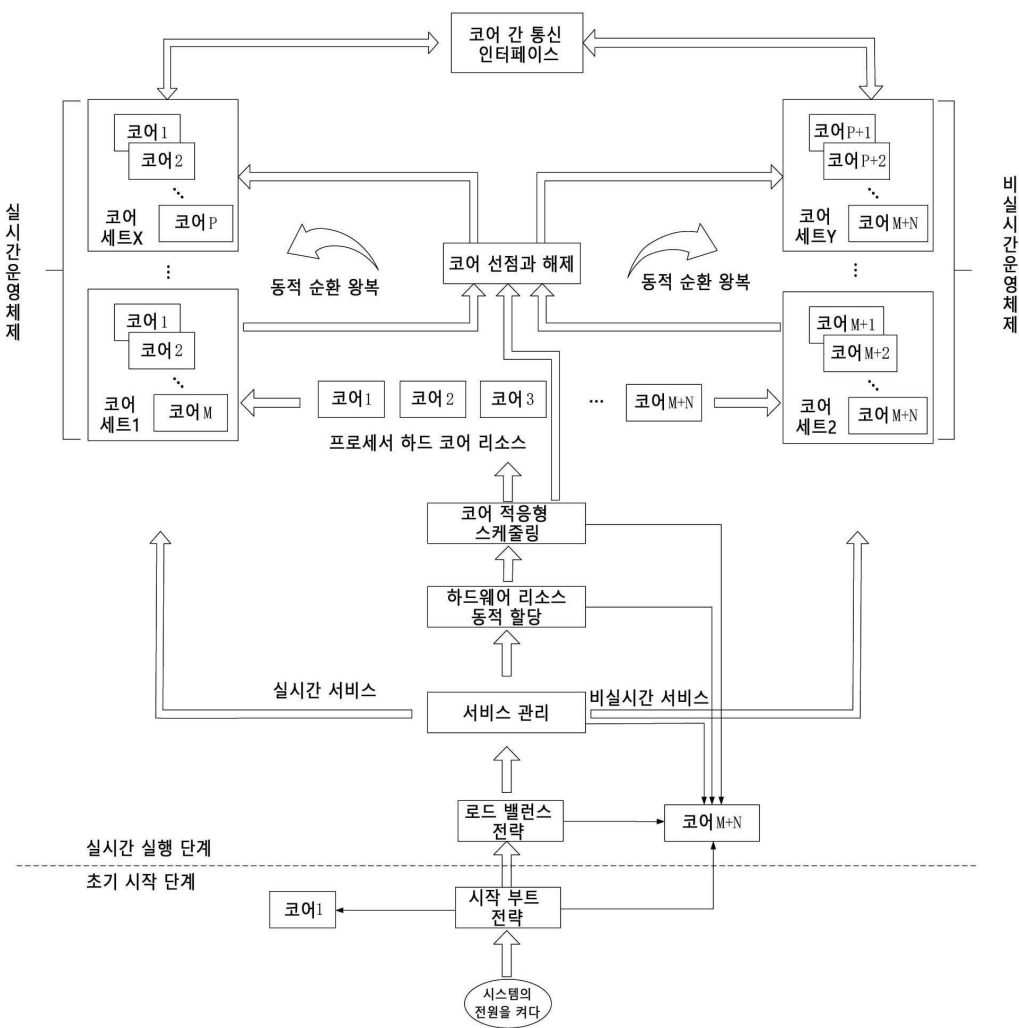
도면5



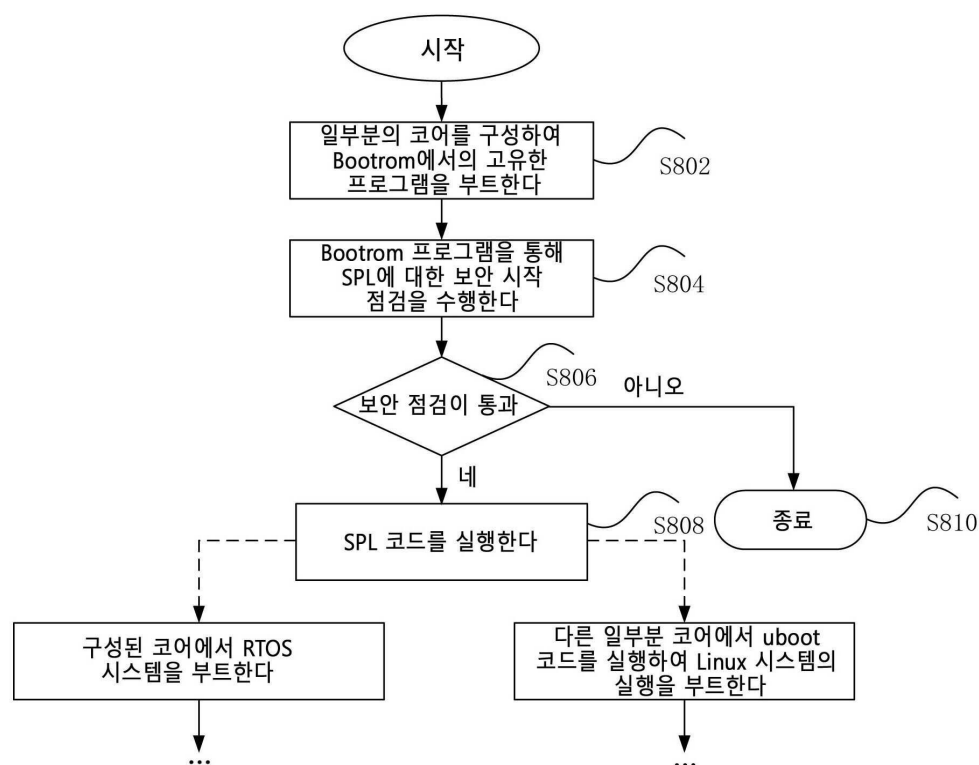
도면6



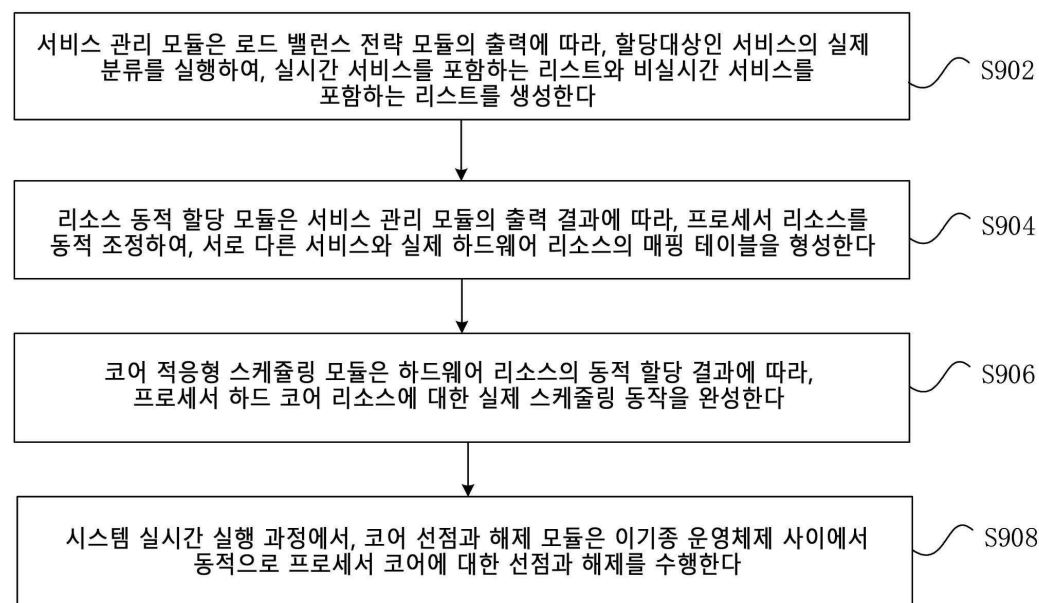
도면7



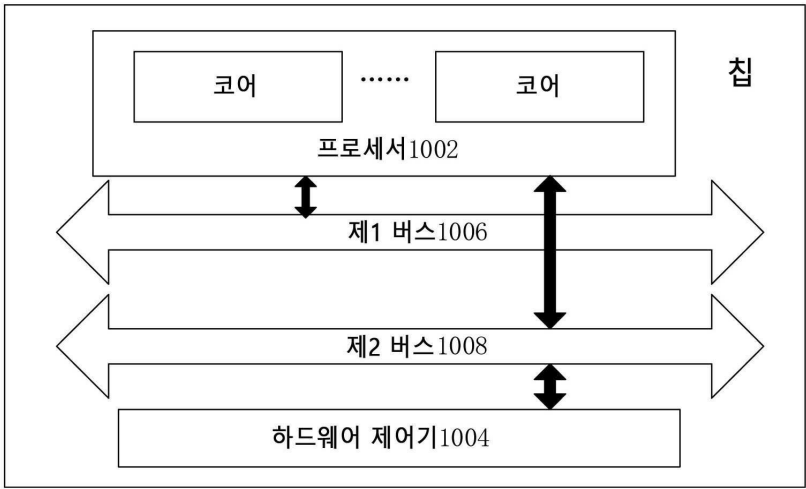
도면8



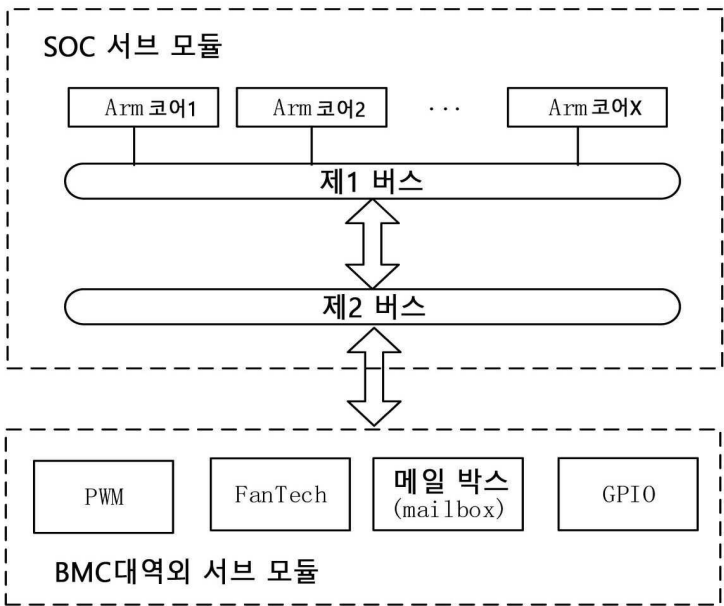
도면9



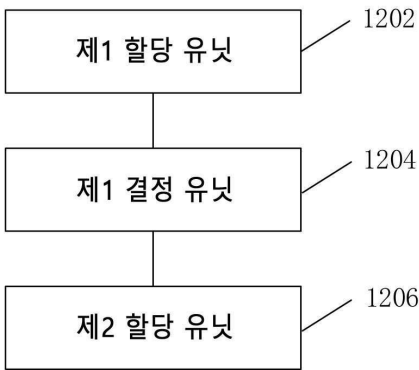
도면10



도면11



도면12



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 청구항 10

【변경전】

제9항에 있어서,

제1 시스템 제어 모듈을 더 포함하고,

제1 시스템 제어 모듈은,

상기 제1 운영체제의 처리 리소스의 리소스 이용율을 검출하고, 상기 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 상기 제1 운영체제에서 상기 제2 운영체제와 상기 리소스 인터랙션을 수행하는 것으로 결정할 경우, 상기 리소스 선점과 해제 모듈에서 상기 코어 간 통신 인터랙션을 통해 상기 제1 인터랙션 요청을 상기 제2 운영체제에 전송하는 것을 트리거하며,

또한, 상기 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 상기 제1 운영체제에서 상기 제2 운영체제의 처리 리소스를 선점하는 것으로 결정하는 단계;

상기 제1 운영체제의 처리 리소스가 유휴 상태이고, 상기 제1 운영체제에서 실행 대상인 서비스가 존재하지 않을 경우, 상기 제1 운영체제에서 상기 제2 운영체제로 처리 리소스를 해제하는 것을 대기하는 것으로 결정하는 단계; 중의 적어도 한 단계를 실행하는 것을 특징으로 하는 임베디드 시스템.

【변경후】

제9항에 있어서,

제1 시스템 제어 모듈을 더 포함하고,

제1 시스템 제어 모듈은,

상기 제1 운영체제의 처리 리소스의 리소스 이용율을 검출하고, 상기 제1 운영체제의 처리 리소스의 리소스 이용율에 따라, 상기 제1 운영체제에서 상기 제2 운영체제와 상기 리소스 인터랙션을 수행하는 것으로 결정할 경우, 상기 리소스 선점과 해제 모듈에서 상기 코어 간 통신 인터랙션을 통해 상기 제1 인터랙션 요청을 상기 제2 운영체제에 전송하는 것을 트리거하며,

또한, 상기 제1 운영체제의 처리 리소스의 리소스 이용율이 제1 이용율 임계값 이상인 경우, 상기 제1 운영체제에서 상기 제2 운영체제의 처리 리소스를 선점하는 것으로 결정하는 단계;

상기 제1 운영체제의 처리 리소스가 유휴 상태이고, 상기 제1 운영체제에서 실행 대상인 서비스가 존재하지 않을 경우, 상기 제1 운영체제에서 상기 제2 운영체제로 처리 리소스를 해제하는 것을 대기하는 것으로 결정하는 단계; 중의 적어도 한 단계를 실행하는 것을 특징으로 하는 임베디드 시스템.