

# 2023년 2학기 스터디그룹 주간학습보고서

그룹 명	MaveOS		
날 짜	11월 1일	시 간	12시 00분 ~ 14시 45분
스터디장소	50주년 기념관 516호	회차	__6 회차 모임
수강과목	운영체제	담당교수	강지훈
스터디	리더: 한아림		
참석자	그룹원: 손효림, 이경화		

## 학습문제

### <CPU 스케줄링3>

- 다중 프로세서 스케줄링
- 실시간 CPU 스케줄링
- 운영체제의 예

## 학습문제 해결과정

### <다중 프로세서 스케줄링>

#### #1. 멀티 프로세서와 시스템 아키텍처

##### 1) 멀티 프로세서의 사용처(Used By System 아키텍처)

: 다중 코어 CPU, 다중 스레드 코어, NUMA 시스템, 이기종 다중 처리

##### 2) 멀티 프로세서 & 스케줄링(스케줄링에 대한 접근)

- 멀티 프로세스 지원을 위한 표준 접근 방식은 대칭 멀티 프로세싱(SMP)이며, 각 프로세서는 스스로 스케줄링할 수 있음.
- 멀티 프로세스 시스템의 스케줄링 방법 중 하나는 단일 코어가 모든 결정을 하는 것
  - 모든 스케줄링 결정 및 I/O 처리, 타 시스템의 활용을 하나의 코어가 수행 --> 나머지 코어는 사용자 코드만 수행
  - 비대칭 멀티 프로세싱은 오직 하나의 코어만 시스템 데이터 구조에 접근 --> 데이터 공유 필요성 배제
  - 단, 모든 결정을 하는 단일 코어가 전체 시스템의 성능을 저하시키는 병목이 발생할 수 있음

##### 3) SMP의 스레드 관리 전략(Managed by Ready Queue)

- 스케줄링 대상 스레드 관리를 위한 전략
  - 모든 스레드가 공통 레디 큐에 위치
  - 각 프로세서는 자신만의 레디 큐 보유

#### 4) 스레드 관리 시 문제(스레드 보존을 위한 방법)

- 각 코어마다 레디 큐가 있는 경우 공용 큐에서 발생하는 성능 문제는 발생하지 않음
  - 자신만의 프로세스별 큐가 있다면 캐시 메모리의 효율적 사용이 가능함
  - 각 코어의 큐마다 부하의 규모가 다를 수 있음 --> 부하 분산 알고리즘이 필요
- 공통 큐는 서로 다른 프로세서가 동일한 스레드를 스케줄하지 않도록 보장해야 함
  - 경쟁 조건으로부터 큐를 보호하기 위해 locking 기능 사용
  - locking 기법은 큐에 대한 모든 액세스에 소유권이 필요하기 때문에 공용 큐 액세스에서 병목 현상 발생 가능

#### 5) 다중 코어 프로세서(시스템 복잡도와 다중 코어)

- 현대의 컴퓨터 하드웨어는 단일 물리 칩 안에 여러 개의 연산 코어를 장착하여 다중 코어 프로세서 형태임
- 각 CPU가 단일 코어를 가지는 다수의 물리 칩으로 구성된 과거의 시스템보다 더 빠르고 적은 전력 소모
- 다중 코어 프로세서는 스케줄링 문제 복잡도를 높임
  - 프로세서가 메모리에 접근할 때 데이터를 기다리며 시간 소모하는 현상 발생
  - 캐시 미스로 인한 Memory Stall 현상 발생 가능
  - 메모리보다 빠른 프로세서의 동작으로 인한 현상

#### 6) 다중 프로세서 스케줄링 → 시간 낭비의 문제

: 프로세서는 메모리의 데이터를 사용할 수 있을 때까지 대기 --> 최대 50%의 시간 낭비 가능

##### 7-1) 다중 프로세서 스케줄링(시간 허비 상황 해결 방법)

- 문제 해결을 위해 HW 설계에서 다중 스레드 처리 코어 구현
  - 하나의 코어에 2개 이상의 하드웨어 스레드 할당
  - 메모리를 기다리는 동안 하드웨어 스레드 중단 시 코어가 다른 스레드로 전환
  - HW 스레드는 명령어 포인터, 레지스터 집합과 같은 구조적 상태를 유지 --> SW 스레드 실행을 위한 논리적 형태

##### 7-2) 다중 프로세서 스케줄링(HW 스레딩과 CMT 기술)

- 하드웨어 스레드 기술은 칩 다중 스레딩(Chip Multithreading) 기술이라고 명명
- 각 코어별로 하드웨어 스레드가 2개씩 할당
  - HW 스레드로 인해 각 코어는 운영체제에게 논리적으로 다중화 --> OS는 CPU 코어가 8개인 것으로 인식하여 스레드 스케줄링 진행
- 인텔의 경우 Hyper-Threading 기술을 통해 일반적으로 코어당 2개의 스레드 지원
  - Oracle Sparc M7 등의 프로세서는 코어당 8개의 스레드 제공

##### 8-1) 프로세서의 다중 스레드화(COARSE-GRAINED 다중 스레딩 기법)

- 메모리 스톨과 같이 긴 대기 시간이 발생할 때까지 스레드가 코어에서 실행
- 프로세서 코어에 다른 스레드가 수행되기 전 명령어 파이프라인이 완전히 정리되어야 함 --> 스레드 간 스위칭 비용 많이 소모
- 새로운 스레드가 시작하면 자신의 명령어들로 파이프라인을 채움

##### 8-2) 프로세서의 다중 스레드화(FINE-GRAINED 다중 스레딩 기법)

- 훨씬 더 세밀한 수준에서 스레드 사이클을 전환
- 일반적으로 명령어 사이클의 경계선에서 발생
- 스레드 교환을 위한 회로를 포함하는 경우가 많음 --> 스레드 간 교환 비용이 적어짐

## #2. SMP 시스템과 부하 분산(Load Balancing)

### 1) 부하 분산(Load Balancing)(SMP 시스템에서의 부담 분산 체계)

- 부하 분산은 통상 각 처리기가 실행할 스레드를 위한 자신만의 레디 큐를 가지고 있는 시스템에서만 필요한 기능
- 부하 분산 기술은 SMP 시스템에서 모든 코어 사이에 부하가 고르게 분배되도록 조절함
- SMP 시스템에서 코어가 하나 이상인 것을 최대한 활용하려면 모든 코어에 균등하게 부하(작업 - 결국은 스레드)를 배분하는 것이 매우 중요
  - 부하를 균등하게 배분하면 특정 코어는 처리를 계속하고 있는데 어떤 코어는 유휴 상태인 경우를 방지할 수 있음
  - 어떤 코어는 레디 큐에 대기 스레드가 많은데 어떤 코어의 레디 큐는 텅텅 비어 있는 경우 방지

### 2-1) 부하 분산을 위한 접근 방식 → Push migration

- 특정 태스크가 주기적으로 각 코어의 부하를 검사하고 불균형 상태인 것을 확인하면 과부하인 코어에서 유휴 상태이거나 덜 바쁜 처리기로 스레드를 이동시키는 방법
- 일반적으로 부하 정도는 레디 큐에서 대기하고 있는 스레드의 개수로 판단하며, 알고리즘에 따라 어떤 스레드를 옮길지를 정하는 방식은 상대적으로 다를 수 있음
- 이동하는 부하를 할당받는 코어 --> 가장 부하가 낮은 코어

### 2-2) 부하 분산을 위한 접근 방식 → Pull migration

- 유휴 코어가 바쁜 코어의 레디 큐에서 대기 중인 스레드를 가져옴

## #3. 다중 프로세서 스케줄링과 프로세서 선호도(Affinity)

### 1) OS 스케줄링과 프로세서 선호도(프로세서 선호도의 형태)

- 약한 선호도(Soft affinity)
  - OS가 프로세스를 특정 코어에서 실행하도록 노력은 하지만 보장하지는 않는 경우
  - 부하 분산이나 선점 등 다양한 이유로 이는 어겨질 수 있음
- 강한 선호도(Hard affinity)
  - 강한 선호도를 제공하는 시스템은 일반적으로 프로세스가 자신이 실행될 코어 집합을 명시할 수 있는 시스템 콜을 제공
  - 리눅스는 약한 선호도를 사용하지만 sched-setaffinity() 시스템 콜을 지원 --> 스레드가 실행할 수 있는 CPU 집합 지정 가능


### 2) NUMA와 프로세서 선호도(아키텍처와 스케줄링의 상호연관성)

- NUMA(non-uniform memory access) 아키텍처 : 각 코어마다 메모리 컨트롤러가 있고 자신의 메모리 영역이 있을 때 코어는 메모리 영역에 존재하는 데이터에 더 빨리 접근
- OS가 NUMA 시스템을 인식하면 스레드가 스케줄링 된 CPU에 해당 하는 메모리를 사용할 수 있도록 할당하여 빠른 메모리 접근을 제공할 수 있음

## #4. 실시간 CPU 스케줄링

### 1) 실시간 OS CPU 스케줄링 : 리얼 타임에 대한 시스템 유형

- Soft real-time system : 기본적으로 데드라인을 보장하기 위해 스케줄러는 노력하지만 작업이 데드라인 이내에 완료하는 것을 보장하지는 않음
- Hard real-time system : 데드라인을 엄격하게 보장하는 것을 목적으로 하며, 데드라인 보장을 위해서는 인터럽트 처리조차도 지연 처리하는 경우도 있음

학습성찰	
학습내용 이해도	95 %
학습활동 돌아보기 (좋았던 점, 아쉬운 점 활동모습 사진 추가)	<p>- 한아림 이번 주차 스터디에서는 다중 프로세서 스케줄링에 대한 보다 더 심도있는 학습을 진행하였다. 스레드의 병렬 처리와 관련된 부하 공유(Load Sharing)과 연산 코어 작용에 대해 학습하며 운영체제의 스케줄링 문제에 대해 접할 수 있었다. 또 다양한 시스템 아키텍처에 대한 내용도 학습하였는데, 그 중에서 다중 코어 CPU, 다중 스레드 코어와 NUMA 시스템, 그리고 이기종 다중 처리에 대한 학습을 수행하였다. 이번 스터디를 통해 멀티 프로세스 시스템의 CPU 스케줄링 방법과 asymmetric multiprocessing 에 대한 지식을 얻을 수 있었다. 다음 시간에는 오늘 배운 내용과 새로운 내용을 결합하여 실습을 해보고자 한다.</p> <p>- 손효림 이번 주는 다중 프로세서 스케줄링과 실시간 CPU 스케줄링에 대해 학습하였다. 교수님께서 기말고사에 들어가는 범위는 아니라고 하셨지만 지적 호기심 탐구 역시 목적으로 하기에 스터디를 변동없이 진행하였다. 이번에 탐구한 내용 중에는 부하균등과 프로세서 선호도 사이에서 이점이 상쇄되는 상충되는 특성이 있는 것이 흥미로웠다. 그리고 얼마 전 중간고사를 응시했는데 스터디가 도움이 많이 되었다. 정말 유익한 점이 많다는 생각이 들었다.</p> <p>- 이경화 이번 스터디는 크게 다중 프로세서 스케줄링, 실시간 CPU 스케줄링, 그리고 운영체제의 예를 주제로 하였다. 기억에 남는 키워드로는 멀티 프로세서, SMP, 하드웨어 스레드, 부하 분산, 프로세스 선호도(Affinity) 등이 있다. 지지난 주차부터 CPU 스케줄링을 주제로 굉장히 많은 내용을 다루었는데, 이제 이 주제를 다루는 마지막 부분에 오니 CPU 스케줄링에 대한 전반적인 내용을 한번씩 다룬 것 같다. 강의 시간에 설명을 듣고, 스터디를 하며 복습하고, 시험 공부를 하느라 또 공부했지만 워낙 내용이 방대한지라 아직도 CPU 스케줄링이 머릿속에 명확하게 이해가 되지는 않은 것 같다. 이 부분에 대해 조금 더 시간을 투자해서 복습을 하고 핵심을 정리하는 작업이 필요하겠다는 생각이 든다.</p> 
다음 학습계획	일정 : 11월 8일 12:00~14:45 (50주년기념관 516호)