CDA0017: Operating Systems

Donghyun Kang (donghyun@changwon.ac.kr)

NOSLab (https://noslab.github.io)

Changwon National University

프로세스에 대한 정보 없이 스케줄링 한다면?

멀티 레벨 피드백 큐 (MLFQ)

- 멀티 레벨 피드백 큐(Multi-level Feedback Queue, MLFQ)
  - 짧은 작업을 먼저 수행함으로써 반환 시간 최적화
    - SJF, STCF의 단점 보완
  - 응답 시간 최적화
    - RR의 단점 보완

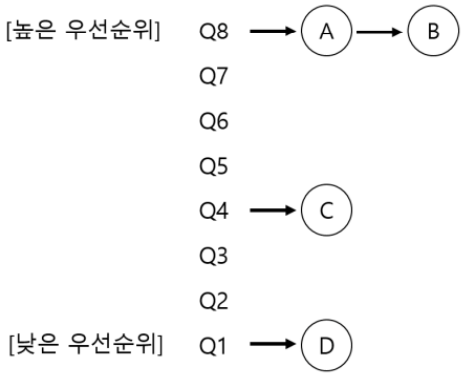
멀티 레벨 피드백 큐 (MLFQ)

- 기본 규칙
  - 여러 개의 큐로 구성
  - 큐는 각각 다른 우선 순위 (Priority)을 가짐: **우선 순위 큐**
  - 큐 내부는 RR 방식으로 동작
  - 생성된 프로세스는 우선 순위 큐 중 하나에 배치됨
  - 높은 우선 순위 큐에 포함된 프로세스 순서로 다음 실행할 프로세스 선택

규칙 1: Priority(A) > Priority(B) 이면, A가 실행됨 (B는 대기)
규칙 2: Priority(A) = Priority(B) 이면, A와 B는 RR 방식으로 실행됨

우선 순위 부여 방식

- 각 프로세스의 특성에 따라 우선 순위 부여
  - 프로세스 A, B는 높은 우선순위 작업
  - 프로세스 C, D는 중간 또는 낮은 우선순위 작업



〈그림 11.1〉 **MLFQ** 예

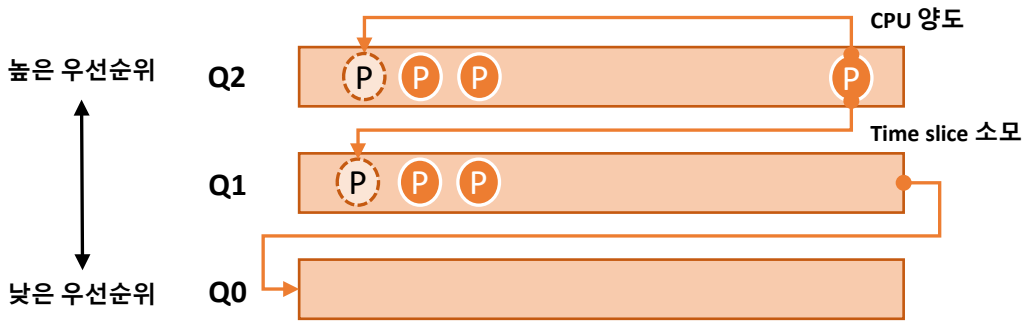
C, D는 언제 실행?

MLFQ 시도 1: 우선순위의 변경

- 워크로드의 특성 반영
  - 프로세스가 짧은 실행 시간을 갖고 CPU을 양보하는 **대화형 워크로드**
    - 키보드 입력을 기다리는 프로세스
  - 응답시간이 중요하지 않으며 오랜 시간 동안 CPU을 집중적으로 사용하는 **배치(batch)형 워크로드**
    - 수학적 연산
    - 인공지능 CNN, RNN

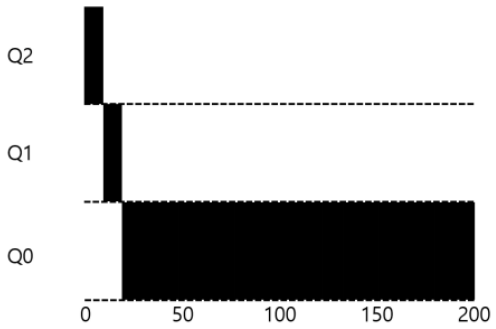
MLFQ 시도 1: 우선순위의 변경

- 우선순위 조정 알고리즘
  - 프로세스가 생성되면 가장 높은 우선순위 부여
    - CPU 선점 후 주어진 시간 (time slice)을 소모하면 낮은 우선순위로 이동
    - 주어진 시간을 소모하지 않고 CPU을 양도하면 같은 우선순위 유지



MLFQ 시도 1: 우선순위의 변경

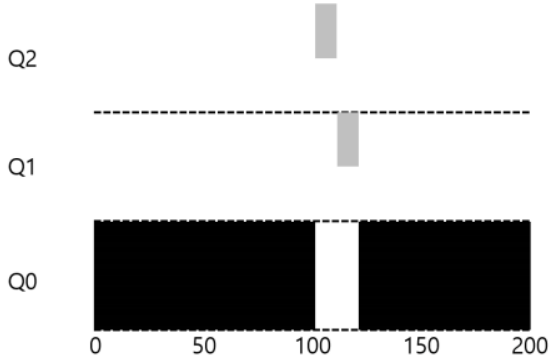
- 오래 실행되는 CPU 워크로드의 경우



〈그림 11.2〉 긴 실행 시간을 가진 작업의 우선순위 변화

MLFQ 시도 1: 우선순위의 변경

- 오래 실행되는 CPU 워크로드와 대화형 워크로드 2개가 존재하는 경우
  - 검정색: 0ms에 도착하고 180ms 수행
  - 회색: 100ms에 도착하고 20ms 수행

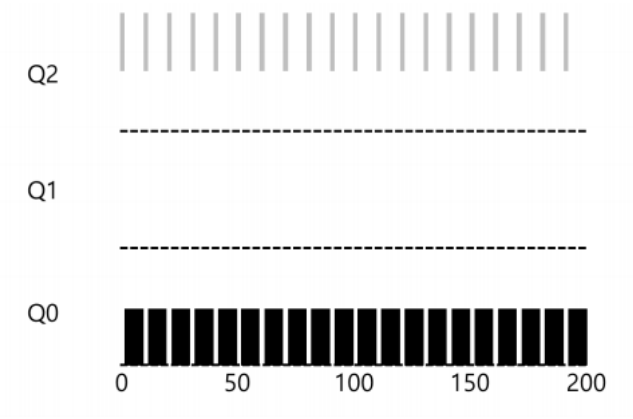


〈그림 11.3〉 대화형 작업이 들어온 경우

짧은 프로세스는 바로 실행된 후 종료
긴 프로세스는 낮은 큐에서 계속해서 수행

MLFQ 시도 1: 우선순위의 변경

- 입출력 작업을 수행하는 경우
  - 검정색: 0ms에 도착하고 180ms 수행
  - 회색: 입출력 수행 전 1ms CPU 작업 수행

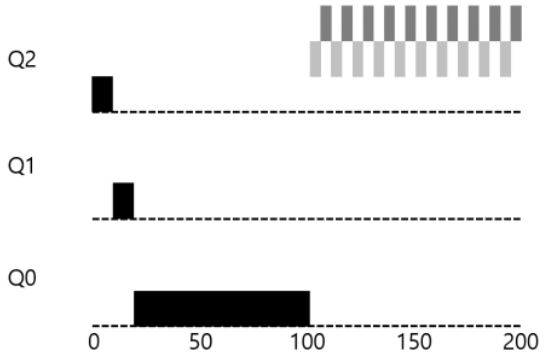


〈그림 11.4〉 입출력-집중 작업과 CPU-집중 작업이 혼합된 워크로드

MLFQ는 완벽한가?

MLFQ 시도 1: 우선순위의 변경

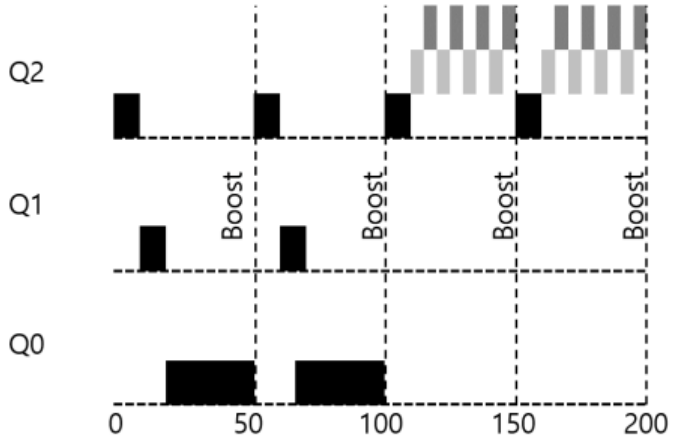
- 기아 상태(starvation) 발생
  - 오랜 시간동안 CPU을 점유를 한번도 하지 못한 상태



- 만약, 프로세스가 99%의 time slice을 사용하고 CPU을 양도하면 항상 가장 높은 우선순위 사용 가능

MLFQ 시도 2: 우선순위의 상향 조정

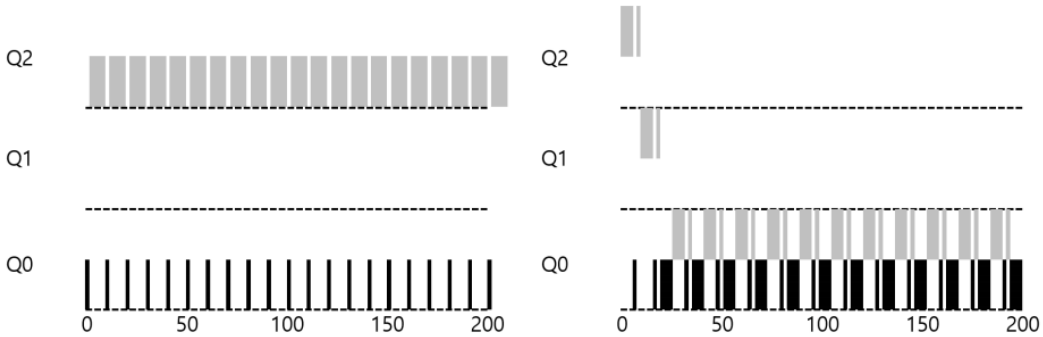
- 우선순위 상향 알고리즘
  - 일정 기간 *S* (부두 상수)가 지나면, 시스템의 모든 프로세스를 최상위 큐로 이동



S값을 얼마로 설정 해야 하는가?

MLFQ 시도 3: 효율적인 시간 측정

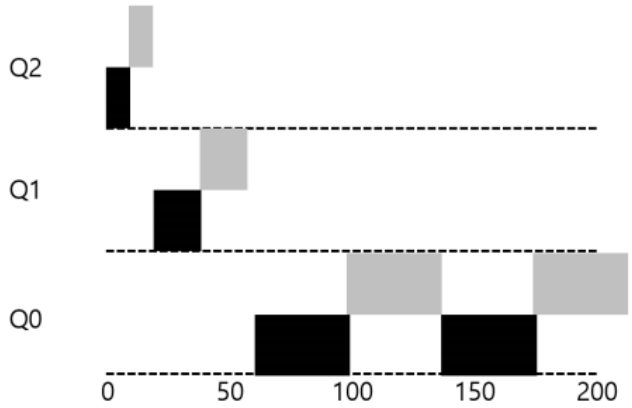
- 시간 측정 알고리즘
  - 주어진 시간을 소진하면 (CPU를 몇 번 양도하였는지 상관없이), 낮은 우선순위로 이동



〈그림 11.6〉 조작에 대한 내성이 없는 경우(좌측)와 있는 경우(우측)

MLFQ 조정과 다른 쟁점들

- MLFQ 동작을 위한 변수들을 어떻게 설정하는가?
  - 큐의 개수
  - 큐당 프로세스의 time slice
  - 우선순위 상향 조정 시간 (*S*: 부두 상수)



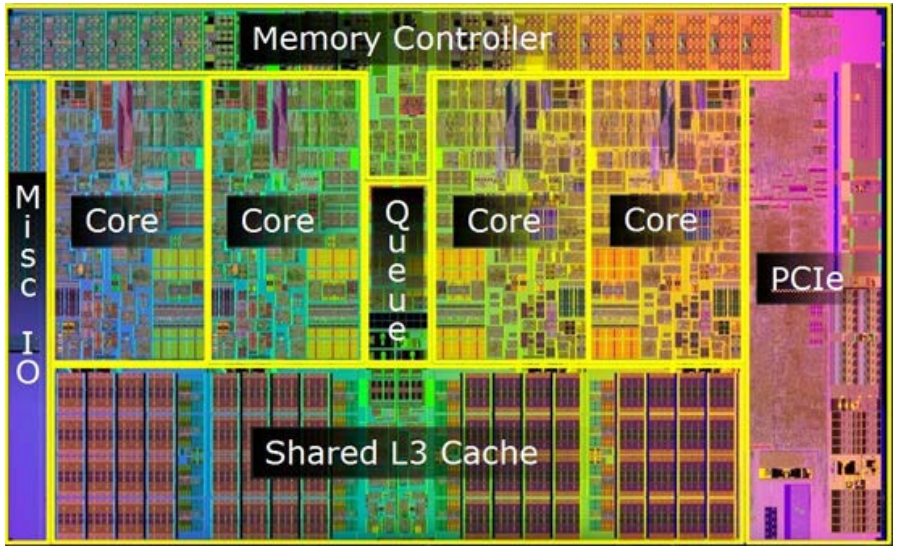
〈그림 11.7〉 낮은 우선순위, 더 긴 할당 시간

MLFQ 정리

- 규칙 1 : 우선순위 (A)> 우선순위 (B) 일 경우, A가 실행
- 규칙 2 : 우선순위 (A) = 우선순위 (B), A와 B는 RR 방식으로 실행
- 규칙 3 : 작업이 시스템에 들어가면 최상위 큐에 배치
- 규칙 4 : 작업이 지정된 단계에서 배정받은 시간을 소진하면 (CPU를 포기한 횟수와 상관없이), 낮은 우선순위로 작업 이동
- 규칙 5 : 일정 주기 S 가 지난 후, 시스템의 모든 작업을 최상위 큐로 이동

MLFQ

MLFQ는 완벽한가?


Memory Controller
Misc IO
Core
Core
Queue
Core
Core
PCIe
Shared L3 Cache

멀티 프로세서?

캐시 (cache)

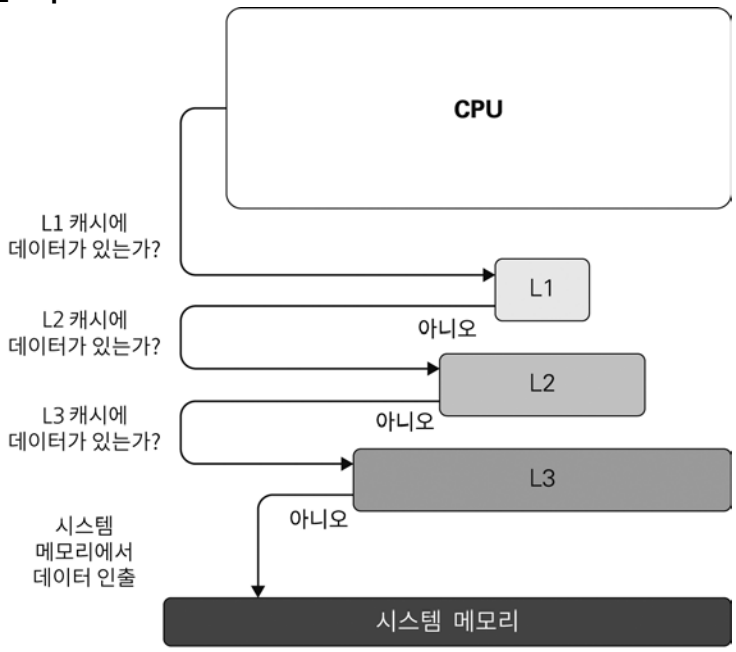
- CPU와 시스템 메모리 (즉, 메인 메모리) 사이에 있는 고속 메모리 블록
  - 메인 메모리에서 자주 사용되는 데이터의 복사본 저장

- 참조 지역성 (Locality)
  - 시간 지역성 (temporal locality)
    - 현재 접근된 동일한 데이터는 가까운 미래에 다시 접근됨
  - 공간 지역성 (spatial locality)
    - 메모리 위치는 순차적으로 접근됨

- 캐시 적중 (cache hit) vs. 캐시 미스 (cache miss)

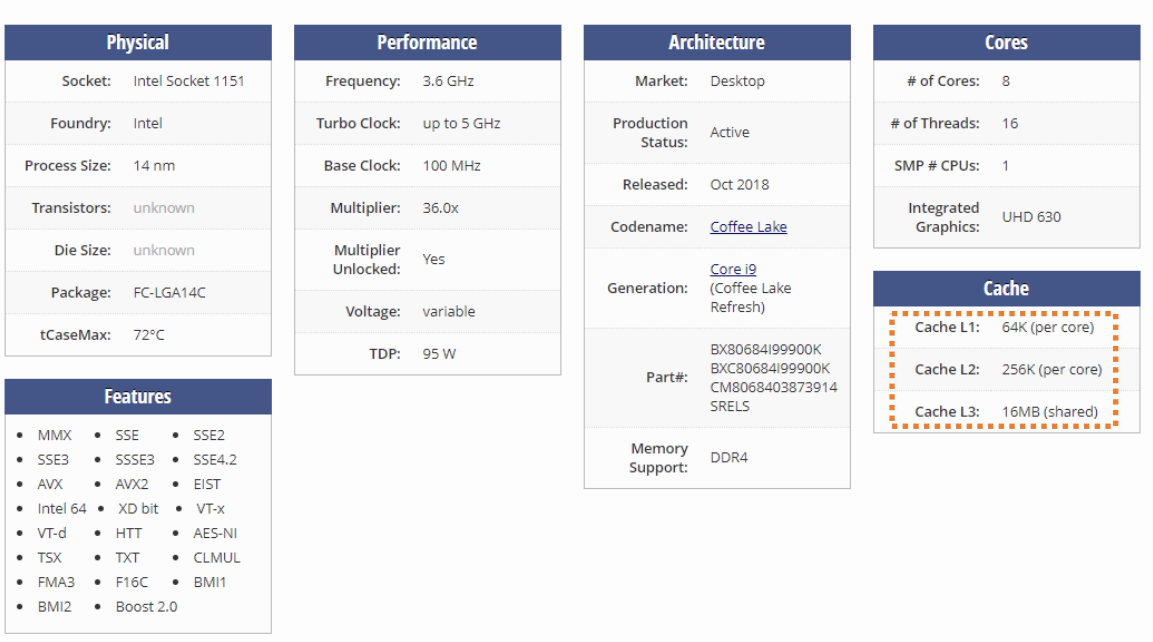
캐시 계층 구조

- 현대 컴퓨터 구조에서, 캐시는 CPU 자체와 동일한 실리콘에 포함됨
  - 가장 빠른 메모리
    - SRAM 방식 채택
    - 작은 양의 크기로 빠른 주소 접근 가능
  - 캐시 크기를 확장하기 위해 계층 구조 선택
    - L1, L2, L3 캐시 (SMP 시스템 구조)
      - L1, L2는 각 core 별로 존재
      - L3는 모든 core가 공유함



실제 CPU 캐시의 크기 확인

- Intel Core i9-9900K



Physical

Socket:	Intel Socket 1151
Foundry:	Intel
Process Size:	14 nm
Transistors:	unknown
Die Size:	unknown
Package:	FC-LGA14C
tCaseMax:	72°C

Performance

Frequency:	3.6 GHz
Turbo Clock:	up to 5 GHz
Base Clock:	100 MHz
Multiplier:	36.0x
Multiplier Unlocked:	Yes
Voltage:	variable
TDP:	95 W

Architecture

Market:	Desktop
Production Status:	Active
Released:	Oct 2018
Codename:	Coffee Lake
Generation:	Core i9 (Coffee Lake Refresh)
Part#:	BX80684I99900K BXC80684I99900K CM8068403873914 SRELS
Memory Support:	DDR4

Cores

# of Cores:	8
# of Threads:	16
SMP # CPUs:	1
Integrated Graphics:	UHD 630

Cache

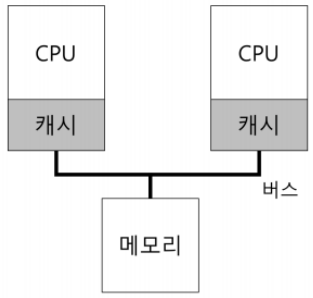
Cache L1:	64K (per core)
Cache L2:	256K (per core)
Cache L3:	16MB (shared)

Features

- MMX • SSE • SSE2
- SSE3 • SSSE3 • SSE4.2
- AVX • AVX2 • EIST
- Intel 64 • XD bit • VT-x
- VT-d • HTT • AES-NI
- TSX • TXT • CLMUL
- FMA3 • F16C • BMI1
- BMI2 • Boost 2.0

멀티 프로세서와 캐시

- 여러 프로세서가 존재하고 하나의 공유 메인 메모리가 있는 경우
  - CPU 1에서 실행 중인 프로그램이 주소 A를 (D 값을) 읽음
  - 주소 A의 값 변경
  - 운영체제가 프로그램의 실행을 중단하고 CPU 2로 이동
  - 프로그램은 CPU2에서 주소 A의 값을 다시 읽음

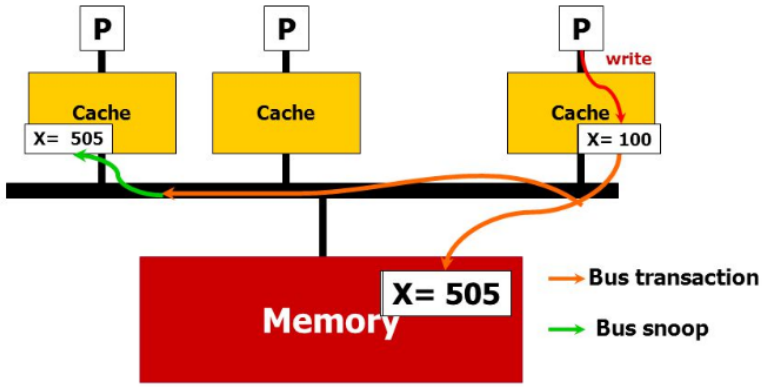


〈그림 13.2〉 메모리를 공유하는 캐시가 장착된 두 개의 CPU

캐시 일관성 (cache coherence) 문제 발생

캐시 일관성 문제 해결 방법

- 스누핑 (snooping) 프로토콜
  - 하드웨어가 메모리 주소를 계속 감시
  - 여러 개의 프로세스들이 공유하고 있는 메모리를 갱신할 때에는 모든 CPU에 항상 공유
  - 상황에 따라, 자신의 복사본을 무효화(invalidate) 시키거나 (즉, 자신의 캐시에서 삭제) 갱신(새로운 값을 캐시에 기록)



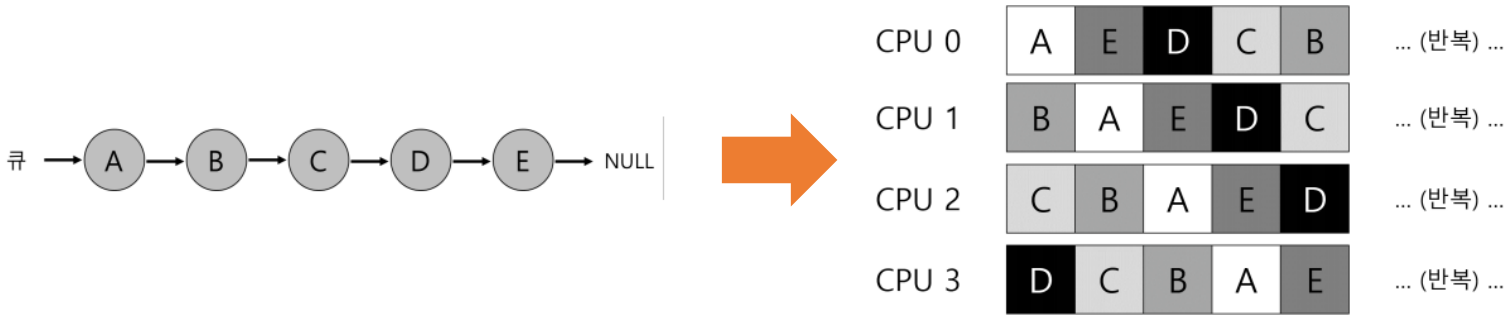
캐시 친화성 (affinity)

- 프로세스가 실행되면 해당 CPU 캐시와 TLB에 프로세스의 상태 정보를 저장함
  - 다음 프로세스 실행 시 동일한 CPU에서 실행되는 것이 유리함
  - 매번 다른 CPU에서 실행되면 실행할 때마다 필요한 정보를 캐시에 다시 로드 (load)

멀티프로세서 스케줄러는 스케줄링 결정을 내릴 때 캐시 친화성을 고려 해야함

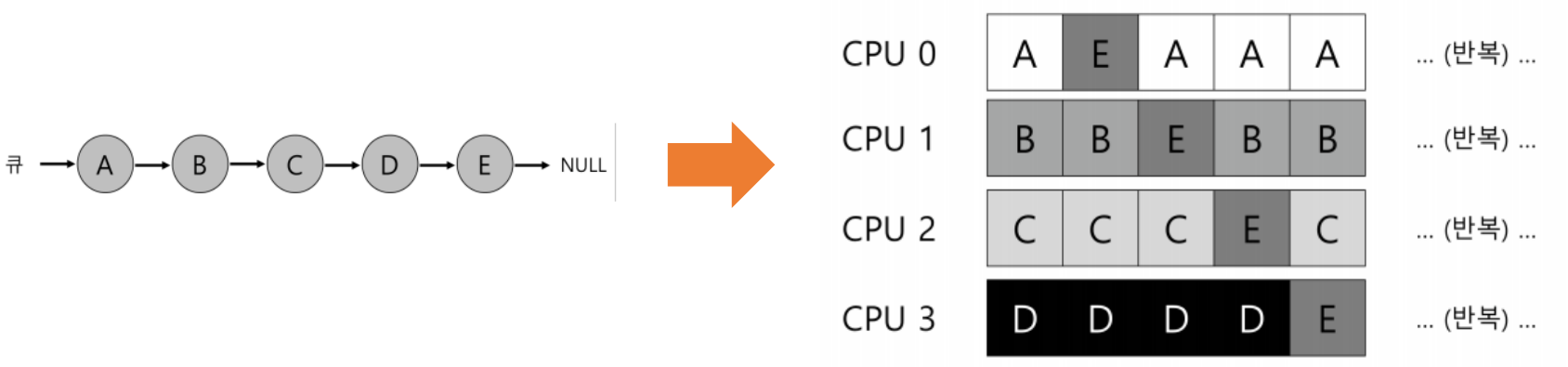
단일 큐 스케줄링 (SQMS)

- Single queue multiprocessor scheduling (SQMS)
  - 1개의 스케줄링 큐로 구성
    - 간단히 구현 가능
    - 확장성(scalability) 이슈
      - 큐의 자료에 접근할 때 lock contention 발생
    - 캐시 친화성 취약



단일 큐 스케줄링 (SQMS)

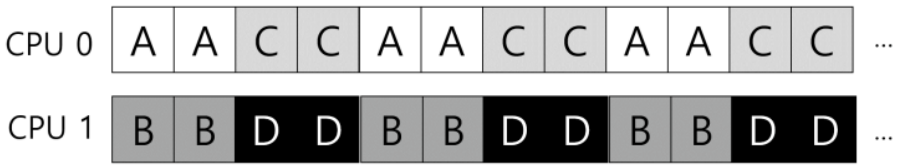
- 가능한 한 프로세스가 동일한 CPU에서 재실행될 수 있도록 배치함



멀티 큐 스케줄링 (MQMS)

- Multi-queue multiprocessor scheduling (MQMS)
  - 여러 개의 스케줄링 큐로 구성
    - 프로세스가 도착하면 하나의 스케줄링 큐에 배치

    - 큐 스케줄링 정책에 따라 각 CPU는 프로세스 스케줄링

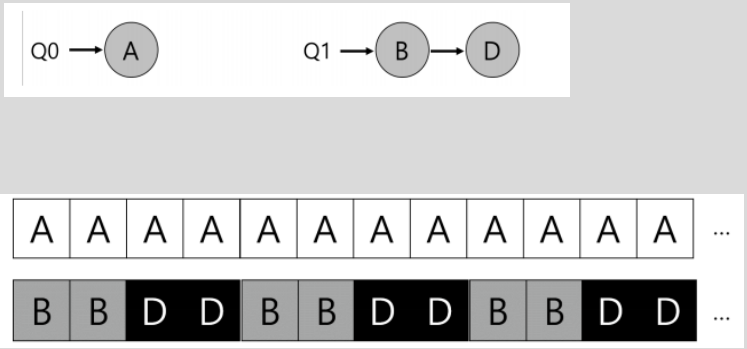


MQMS의 문제는?

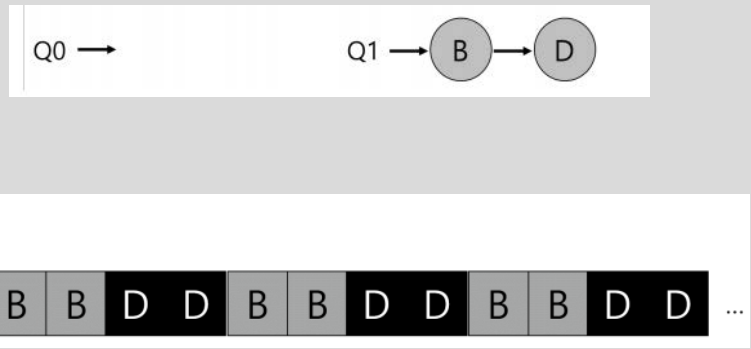
멀티 큐 스케줄링 (MQMS)

- 워크로드의 불균형(load imbalance)

예제 1

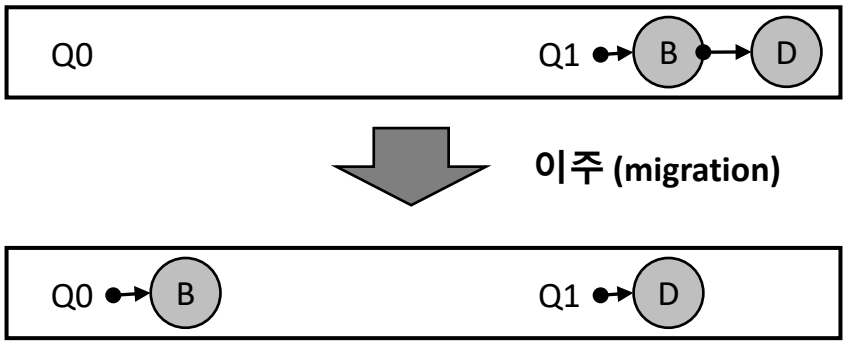


예제 2



멀티 큐 스케줄링 (MQMS)

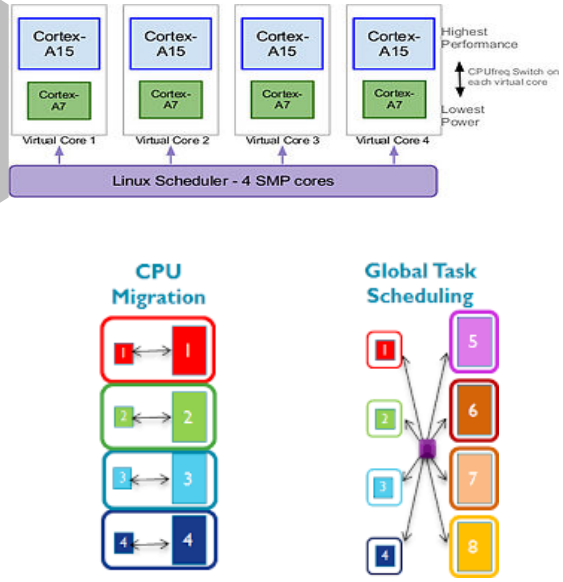
- 이주(migration)
  - 프로세스를 한 CPU에서 다른 CPU로 이주



갑자기 Q0에 프로세스가 많아진다면, B는?

멀티 큐 스케줄링 (MQMS)

- big.LITTLE에서의 프로세스 이주



Q&A