CDA0017: Operating Systems

Donghyun Kang (donghyun@changwon.ac.kr)

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

Changwon National University

Scheduler

- 프로세스 스케줄링의 원칙
  - 모든 프로세스는 공정 해야함 (Fairness)
  - 모든 프로세스의 효율성
    - 작업 (Job) 종료 시간?
    - 처음 CPU을 선점하는 시간?

스케줄러 이해를 위한 가정

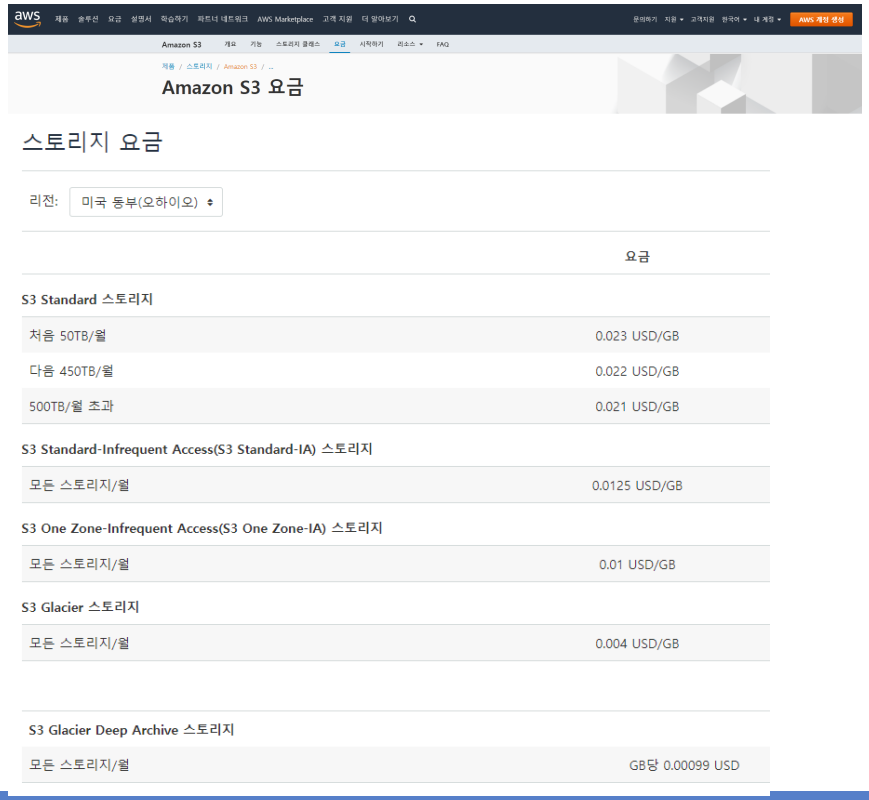
1. 모든 작업은 같은 시간 동안 실행된다.
2. 모든 작업은 동시에 도착한다.
3. 각 작업은 시작되면 완료될 때까지 실행된다
4. 모든 작업은 CPU만 사용한다 (즉, 입출력을 수행하지 않는다).
5. 각 작업의 실행 시간은 사전에 알려져 있다.

스케줄러 평가 항목

- **반환 시간(turnaround time)**
  - 작업이 CPU을 점유하고 수행한 전체 시간

- **공정성 (fairness)**

https://aws.amazon.com/ko/s3/pricing/



Amazon S3 요금

스토리지 요금

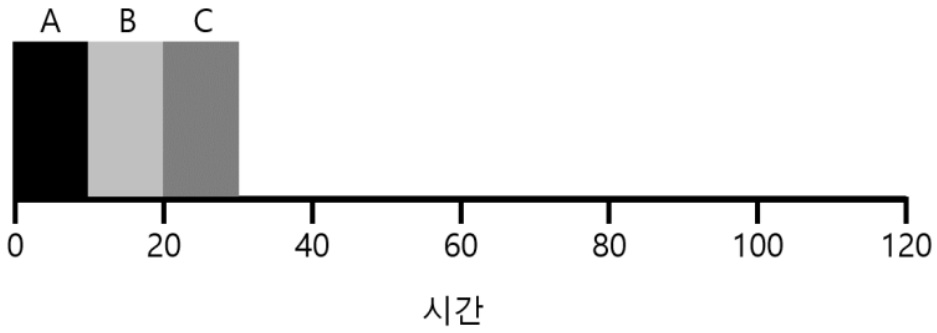
리전: 미국 동부(오하이오)

	요금
S3 Standard 스토리지	
처음 50TB/월	0.023 USD/GB
다음 450TB/월	0.022 USD/GB
500TB/월 초과	0.021 USD/GB
S3 Standard-Infrequent Access(S3 Standard-IA) 스토리지	
모든 스토리지/월	0.0125 USD/GB
S3 One Zone-Infrequent Access(S3 One Zone-IA) 스토리지	
모든 스토리지/월	0.01 USD/GB
S3 Glacier 스토리지	
모든 스토리지/월	0.004 USD/GB
S3 Glacier Deep Archive 스토리지	
모든 스토리지/월	GB당 0.00099 USD

선입선출

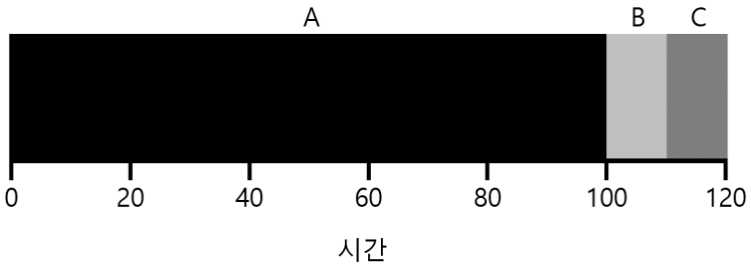
- 선입선출(First In First Out, **FIFO**)

- 스케줄링 시뮬레이션1
  1. 작업 A, B, C가 동시에 생성되어 CPU 선점을 대기함
  2. 각 작업은 각각 10초 동안 CPU을 점유함



선입선출

- 스케줄링 시뮬레이션2
  1. 작업 A, B, C가 동시에 생성되어 CPU 선점을 대기함
  2. **A는 100초 동안 CPU을 점유하고 B, C는 10초 동안 CPU을 점유함**



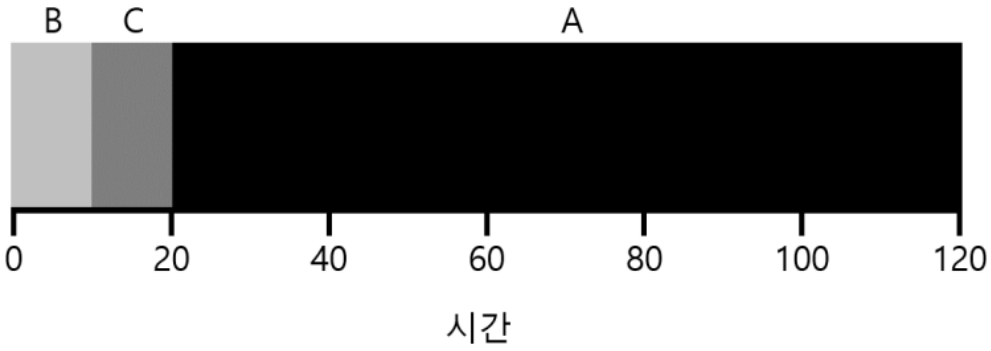
- 평균 반환 시간

합리적인 스케줄러인가?

최단 작업 우선

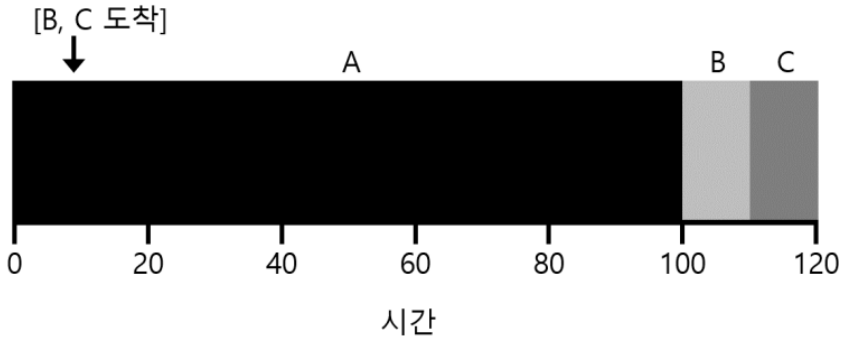
- 최단 작업 우선(Shortest Job First, **SJF**)

- 스케줄링 시뮬레이션2
  1. 작업 A, B, C가 동시에 생성되어 CPU 선점을 대기함
  2. A는 100초 동안 CPU을 점유하고 B, C는 10초 동안 CPU을 점유함



최단 작업 우선

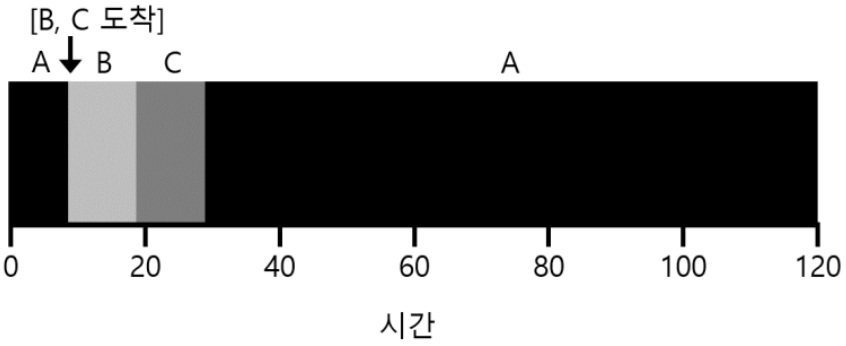
- 스케줄링 시뮬레이션3
  1. **작업 A, B, C가 임의의 시간에 생성되어 CPU 선점을 대기함**
     1. **A는 0초에 생성**
     2. **B, C는 10초에 생성**
  2. A는 100초 동안 CPU을 점유하고 B, C는 10초 동안 CPU을 점유함



최소 잔여시간 우선

- 최단 잔여시간 우선(Shortest Time-to-Completion First, **STCF**)

- 스케줄링 시뮬레이션3
  1. **작업 A, B, C가 임의의 시간에 생성되어 CPU 선점을 대기함**
     1. **A는 0초에 생성**
     2. **B, C는 10초에 생성**
  2. A는 100초 동안 CPU을 점유하고 B, C는 10초 동안 CPU을 점유함



새로운 평가 항목

- **응답 시간(response time)**
  - 작업이 생성 및 도착할 때부터 처음 스케줄 될 때까지의 시간

$$T_{response} = T_{firstrun} - T_{arrival}$$

- 스케줄링 시뮬레이션4
  1. 작업 A, B, C가 임의의 시간에 생성되어 CPU 선점을 대기함
     1. A는 0초에 생성
     2. B, C는 10초에 생성
  2. **A, B, C는 10초 동안 CPU을 점유함**

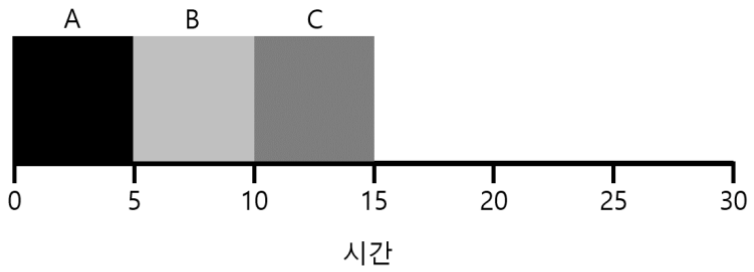
라운드 로빈

- 라운드 로빈(Round-Robin, **RR**)
  - 일정 시간 (*time slice* 또는 *scheduling quantum*) 동안 실행 후 다음 작업 수행

- 스케줄링 시뮬레이션5
  1. **작업 A, B, C가 동시에 생성되어 CPU 선점을 대기함**
  2. **A, B, C는 5초 동안 CPU을 점유함**

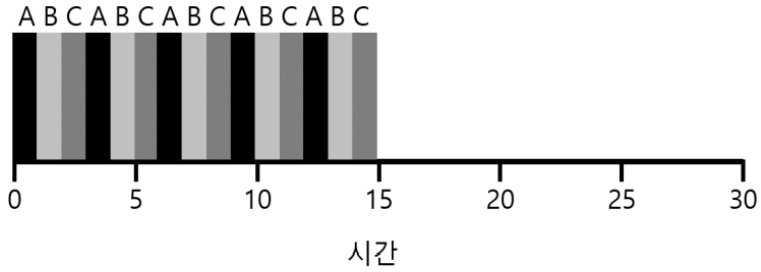
라운드 로빈 vs. SJF

- 응답 시간 기준



〈그림 10.6〉 다시 SJF 스케줄링(응답 시간은 좋지 않음)

평균 응답 시간: $\frac{(0+5+10)}{3} = 5$



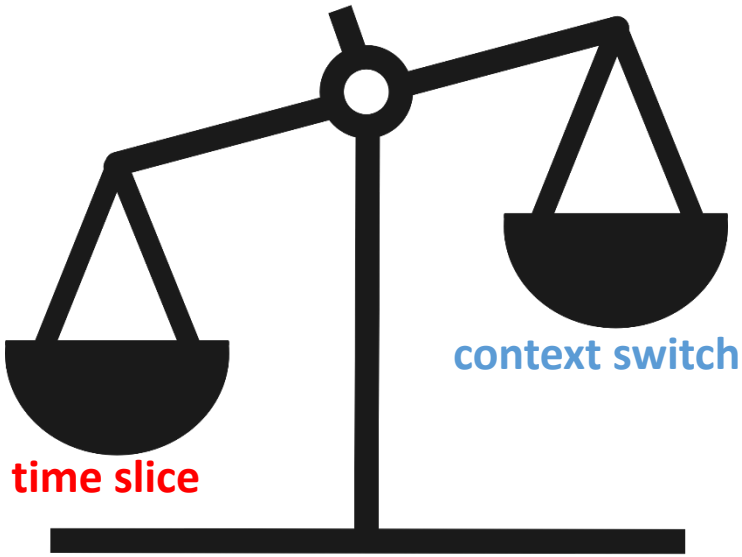
〈그림 10.7〉 라운드 로빈(응답 시간이 좋음)

평균 응답 시간: $\frac{(0+1+2)}{3} = 1$

스케줄링의 오버헤드

- 문맥 교환 (context switch)
  - 현재 수행 중인 프로세스의 정보를 임시 저장함
    - CPU 레지스터 정보, TLB 캐시, 분기 예측, 등.

- 오버헤드 예측
  - If (context switch > time slice) then…
  - If (context switch == time slice) then…
  - If (context switch < time slice) then…



정리

- FIFO, SJF, STCF, RR 스케줄러

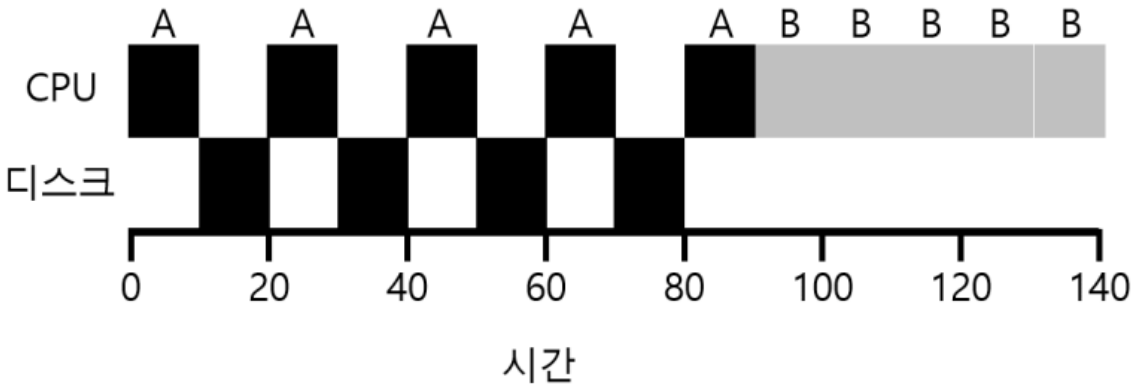
	반환 시간	응답 시간
FIFO		
SJF		
STCF		
RR		

스케줄러 이해를 위한 가정

1. 모든 작업은 같은 시간 동안 실행된다.
2. 모든 작업은 동시에 도착한다.
3. 각 작업은 시작되면 완료될 때까지 실행된다
4. **모든 작업은 CPU만 사용한다 (즉, 입출력을 수행하지 않는다).**
5. **각 작업의 실행 시간은 사전에 알려져 있다.**

입출력 연산의 고려

- 스케줄링 시뮬레이션6
  1. **작업 A, B가 동시에 생성되어 CPU 선점을 대기함**
  2. **A, B는 50msec 동안 CPU을 점유함**
  3. **A는 10msec 실행 후 10mesc의 입출력을 수행함**



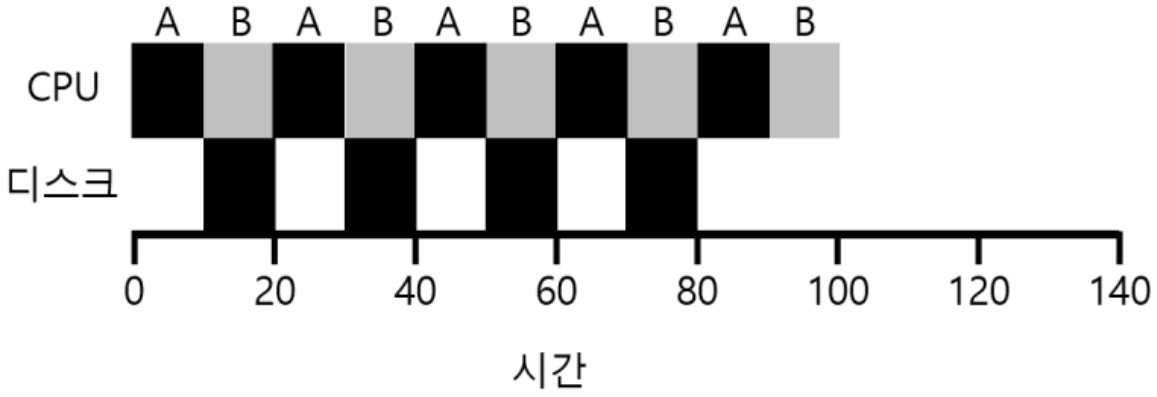
FIFO 스케줄러

입출력 연산의 고려

- 스케줄링 시뮬레이션6 정리
  - A는 10msec 작업 5개 구성됨
  - B는 50mes 작업 1개로 구성됨

- 입출력 연산을 중첩 시키는 STCF

현재 스케줄링 정책은 효율적인가?



스케줄링 관련 최신 연구 동향

[HTML] vSimilar: A high-adaptive VM scheduler based on the **CPU** pool mechanism — [HTML] sciencedirect.com

L Lin, X Liu, R Ma, J Li, D Wang, H Guan - Journal of Systems Architecture, 2019 - Elsevier — Find it @ DGUG

… VM context switching. A single VM context switching operation can use more than 2000 **CPU** cycles, and context switching is the most **CPU**-resource-expensive operation performed by VMM **schedulers** [22]. Shortening VM time …

☆ 99 관련 학술자료 전체 3개의 버전 »

Utilization of Fuzzy Logic in **CPU** Scheduling in Various Computing Environments — [PDF] researchgate.net

B Granam, H ElAarag - Proceedings of the 2019 ACM Southeast …, 2019 - dl.acm.org

… For comparing **CPU schedulers**, several criteria have been suggested. These criteria include [1]: **CPU** utilization, throughput … In section 5, we compare the performance of the fuzzy logic based **CPU** scheduler and other well-known **CPU schedulers** …

☆ 99 관련 학술자료 전체 2개의 버전 »

Smart Scheduler for CUDA Programming in Heterogeneous **CPU**/GPU Environment

NA Khan, MB Latif, N Pervaiz, M Baig… - Proceedings of the 11th …, 2019 - dl.acm.org

… Heterogeneous **CPU**/GPU Environment … heterogeneous CUDA environment which ensures that while the task is fetched into the system, all the nodes participate fully in scheduling, thereby completing the task in less span of time as compared to normal **schedulers** resulting in …

☆ 99 관련 학술자료 »

Influence of Tasks Duration Variability on Task-Based Runtime **Schedulers** — [PDF] ieee.org

O Beaumont, L Eyraud-Dubois… - 2019 IEEE International …, 2019 - ieeexplore.ieee.org

… this variability on the behavior of runtime **schedulers**. We consider a typical HPC node consisting of both multicore and accelerators. We made our experiments on the sirocco platform, which is made of 2 Dodeca-core Intel Xeon E5-2680, providing a total of 24 **CPU** cores, and 4 …

☆ 99 1회 인용 관련 학술자료 전체 4개의 버전 »

Deciphering Predictive **Schedulers** for Heterogeneous-ISA Multicore Architectures — [PDF] virginia.edu

A Prodromou, A Venkat, DM Tullsen - Proceedings of the 10th …, 2019 - dl.acm.org

… investigated to the same extent as single-ISA or **CPU**-GPU heterogeneous architectures. Popcorn Linux imple- ments a scheduler that relies on application instrumentation and generates a mapping of application's functions to cores [10]. 3 Motivation **Schedulers** clearly benefit …

☆ 99 1회 인용 관련 학술자료 전체 4개의 버전 »

Q&A