1주차

1열	운영체제 개요
2열	3/13
Status	완료

문제집 (중간고사 대비)

4장_CPU 스케줄링.pdf

3장_프로세스와 스레드.pdf

1장_운영체제의 개요.pdf

평가 관련 안내

- 1학기 : 중간고사(40%) + 노션 정리(30%) + 논술(30%)
- 2학기 : 미정

1~2주차 : 가볍게 읽고, 내용 이해하기

단, 커널의 모드와 종류(모놀로틱, 마이크로)에 대해서는 정리하기

운영체제 개요

1. 운영체제

1. 컴퓨터 자원을 효율적으로 관리하고 운영하는 시스템 소프트웨어
2. 사용자가 편리하게 컴퓨터를 활용할 수 있는 환경을 제공
3. 목표 : 범용성, 확장성, 적응성, 처리 능력 향상, 응답 시간 단축, 사용의 용이성, 가동성 향상

1.1. 운영체제의 주요 목적

- **사용의 용이성**
  - 하드웨어와 정보를 효율적으로 관리하여 컴퓨터를 편리하게 사용할 수 있도록 지원
- **시스템 성능 향상**
  - **처리 능력 (throughput)**: 일정 시간 동안 컴퓨터가 처리하는 작업량
  - **응답 시간 (turnaround time)**: 사용자가 요청한 작업이 완료되기까지의 시간
  - **사용의 용이성 (availability)**: 시스템 자원의 신속한 제공 여부
  - **신뢰도 (reliability)**: 시스템의 정확한 동작 정도

1.2. 운영체제의 기능

자원 관리와 **시스템 관리**로 구분

- **자원 관리(중요)**: **메모리 관리, 프로세스 관리, 주변 장치 관리, 파일(데이터) 관리**
- **시스템 관리**: 시스템 보호(사용자 권한 관리), 네트워킹(통신), 명령 인터프리터 제공

1.3. 운영체제의 기능 분류

- 감시 프로그램 : 작업 관리, 자원 할당과 회수 등 시스템 전반을 감독하고 제어함
- 작업 제어 프로그램 : 작업의 순서를 정하고 작업 흐름을 통제함

- 데이터 관리 프로그램 : 파일, 데이터의 표준적 관리 및 처리를 담당함

- 참고 : 운영체제의 역할인 것, 역할이 아닌 것

▼ 역할인 것

  - 사용자 인터페이스 제공
  - 주변 장치 관리
  - 자원 분배와 효율적 관리
  - 신뢰성 향상 및 오류 처리

▼ 역할이 아닌 것

  - 원시 프로그램을 기계어로 번역하는 기능 (컴파일러의 역할)
  - 목적 프로그램의 생성 및 연결(링커의 역할)
  - 데이터 압축 및 복원(유틸리티 프로그램 역할)

1.4. 운영체제의 구성 요소와 역할

- 커널, 시스템 호출(system call)

2. 운영체제 연산 (커널과 시스템 호출)

인터럽트 기반(interrupt driven) 동작 (인터럽트가 발생할 때만 작업을 수행)

2.1. 트랩과 시스템 호출

- **트랩(trap)**: 오류 발생 시 또는 운영체제 서비스 요청 시 발생하는 소프트웨어 인터럽트
- **시스템 호출(system call)**: 사용자 프로그램이 운영체제 권한이 필요한 작업을 요청하는 방법
  - 트랩을 통해 실행되거나 시스템 호출 명령을 통해 수행됨

- 운영체제는 모드 비트를 이용하여 **사용자 모드(1)**와 **커널 모드(0)**를 구분

2.2. 사용자 모드와 커널 모드

운영체제는 **사용자 모드(User Mode)**와 **커널 모드(Kernel Mode)** 두 가지 모드를 제공.

운영체제의 보안을 유지하고 시스템의 안정성을 확보하기 위한 중요한 기법

- **사용자 모드(User Mode)**
  - 일반 응용 프로그램이 실행되는 환경
  - 하드웨어 자원에 직접 접근할 수 없음
  - 제한된 명령어만 실행 가능하며, 시스템 호출을 통해 운영체제의 서비스 요청 가능
- **커널 모드(Kernel Mode)**
  - 운영체제의 핵심 기능이 실행되는 환경
  - 하드웨어 자원에 직접 접근 가능
  - 모든 명령어를 실행할 수 있으며, 프로세스 및 메모리 관리, 장치 제어 등을 수행

중요 : 운영체제는 **일반적으로 시스템이 부팅될 때 커널 모드에서 실행**된다. 이후 **응용 프로그램이 실행되면 사용자 모드로 전환**된다. **특정 작업(예: 입출력, 메모리 할당 등)을 수행할 때는 시스템 호출을 통해 다시 커널 모드로 전환**된다. 이를 통해 시스템의 보안을 강화하고 응용 프로그램이 운영체제의 핵심 기능을 침해하지 않도록 보호할 수 있다.

- 참고 : 커널과 쉘

▼ 커널

- 운영체제의 중심부로 하드웨어를 직접 관리하고 제어하는 핵심적 역할을 수행
- 주된 역할:
  - 프로세스 관리(생성, 종료, 스케줄링 등)
  - 메모리 관리(할당, 회수, 가상 메모리 등)

    - 파일 시스템 관리(파일 입출력, 파일 접근 관리 등)
    - 입출력 장치 관리 및 하드웨어 자원 관리
- 커널은 운영체제의 가장 내부에서 작동하여 응용 프로그램과 하드웨어 사이의 중재자 역할을 수행

▼ 쉘

- 쉘은 사용자가 운영체제와 상호작용할 수 있도록 제공하는 **사용자 인터페이스(UI)** 프로그램
- 주된 역할:
    - 사용자 명령어를 해석하여 운영체제의 커널에 전달함.
    - 사용자의 명령을 실행하고 결과를 다시 사용자에게 전달하는 역할을 수행
- 사용자는 직접 커널에 접근할 수 없고, 쉘을 통해 운영체제의 기능을 사용할 수 있음.

3. 운영체제 구조 (Operating System Structures)

3.1. 운영체제의 구조

1. **간단한 구조 (Simple Structure)**
    - 초기 시스템에서 사용
    - MS-DOS, 초기 UNIX와 같은 운영체제에서 계층이 잘 구분되지 않음

2. **계층적 접근 (Layered Approach)**
    - 운영체제를 계층별로 나누어 모듈화
    - 모듈화된 운영체제는 디버깅이 용이하고 유지보수가 쉬움
    - 예: THE 운영체제, OS/2

3. **마이크로커널 (Microkernel)**
    - 운영체제의 핵심 기능을 최소화하고, 나머지는 사용자 공간에서 수행
    - 장점: 확장 용이, 이식성 증가, 보안성 및 신뢰성 향상
    - 예: Mach 운영체제

4. **모놀리식 커널 (Monolithic Kernel)**
    - 운영체제의 모든 핵심 기능이 커널 내부에서 실행됨
    - 커널 내에서 모든 서비스(파일 시스템, 메모리 관리, 장치 드라이버 등)가 포함됨
    - 장점: 실행 속도가 빠르고, 커널 간 통신 오버헤드가 적음
    - 단점: 커널 크기가 커지면 유지보수가 어려워지고, 버그 발생 시 시스템 전체에 영향을 미칠 가능성이 높음
    - 예: Linux, UNIX

4. 운영체제의 발달 과정

운영 방식	특징
일괄처리(배치 시스템)	데이터를 모아 한꺼번에 처리. 작업이 순차적으로 진행. CPU 유휴 시간이 많아 비효율적일 수 있음. 예: 급여 계산
다중 프로그래밍 시스템	하나의 CPU가 여러 프로그램을 동시에 실행. CPU 활용도를 높일 수 있음
시분할 시스템	여러 사용자가 동시에 프로그램을 실행. CPU 시간을 잘게 나누어 사용. 예: 라운드 로빈 방식
다중 처리 시스템	여러 개의 CPU가 하나의 작업을 병렬로 처리. CPU 간 부하 분산이 중요
실시간 처리 시스템	즉각적인 데이터 처리 필요. 응답 시간이 짧아야 함. 예: 레이더 추적기, 은행 좌석 예약 시스템
다중 모드 처리 시스템	일괄 처리, 시분할, 다중 처리, 실시간 처리 시스템을 조합하여 제공. 유연성이 높음
분산 처리 시스템	여러 컴퓨터가 네트워크를 통해 작업을 공유. 독립적으로 동작하면서도 협력하여 전체 성능 향상. 예: 클라우드 컴퓨팅

▼ 해설(1~20번)

1. 2018년 2회-6번

운영체제의 성능 판단 요소와 거리가 먼 것은?

- 정답: ② **비용**
- 해설:
  - 성능 평가 요소는 처리능력, 신뢰도, 사용가능도 등이 있음.
  - 비용은 경제적 요소로, 운영체제 성능 요소에 포함되지 않음.

2번 (2018년 2회-8번)

- 정답: ③ **Real-time processing system (실시간 처리 시스템)**
- 해설:
  - 비행기 제어, 교통 제어와 같이 반드시 정해진 시간 내 처리가 필요한 작업에 적합함.

3번 (2018년 2회-13번)

- 정답: ② **사용자 인터페이스**
- 해설:
  - 사용자 인터페이스는 주로 쉘(shell)이 제공하며, 커널의 핵심 기능이 아님.

4번 (2018년 2회-20번)

- 정답: ④ **데이터 관리 프로그램**
- 해설:
  - 데이터 관리 프로그램은 주기억장치와 보조기억장치 간 데이터 전송, 파일 관리 등의 역할을 함.

5번 (2018년 1회-8번)

- 정답: ③ **Real Time Processing (실시간 처리 시스템)**
- 해설:
  - 정해진 시간 내 처리해야 하는 작업은 실시간 처리 방식이 적합함.

6번 (2018년 1회-15번)

- 정답: ③ **Time Sharing System (시분할 시스템)**
- 해설:
  - CPU 시간을 작은 단위(time slice)로 나누어 각 사용자에게 균등하게 할당하는 방식임.

7번 (2017년 3회-12번)

- 정답: ③ **운영체제의 종류로는 매크로 프로세서, 어셈블러, 컴파일러 등이 있다.**
- 해설:
  - 매크로 프로세서, 어셈블러, 컴파일러는 언어 번역 프로그램이며 운영체제가 아님.

8번 (2017년 2회-5번)

- 정답: ④ **목적 프로그램과 라이브러리, 로드 모듈을 연결하여 실행 가능한 로드 모듈을 만든다.**
- 해설:
  - 해당 기능은 링커(linker)의 역할이며, 운영체제의 기능이 아님.

9번 (2017년 2회-14번)

- 정답: ④ **서비스 프로그램**
- 해설:
  - 서비스 프로그램은 시스템 유지보수 프로그램으로, 제어 프로그램과는 거리가 멈.

10번 (2017년 2회-16번)

- 정답: ② **운영체제는 사용자와 컴퓨터 시스템 사이에서 제어해주는 하드웨어이다.**

- 해설:
  - 운영체제는 하드웨어가 아니라 소프트웨어임.

11번 (2017년 2회-17번)

- 정답: ② **Availability (사용 가능도)**
- 해설:
  - 여러 사용자 요구를 신속하게 지원할 수 있는 정도를 나타내는 요소가 사용 가능도임.

14번 (2017년 1회-8번)

- 정답: ① **Availability (사용 가능도)**
- 해설:
  - 시스템 자원을 얼마나 신속하고 충분히 지원할 수 있는지 나타내는 사용 가능도가 맞음.

15번 (2016년 3회-4번)

- 정답: ③ **사용자가 작성한 원시 프로그램을 기계 언어로 번역시키는 기능**
- 해설:
  - 원시 프로그램을 기계 언어로 번역하는 것은 운영체제가 아닌 컴파일러의 역할임.

16번 (2016년 3회-17번)

- 정답: ④ **운영체제의 종류에는 UNIX, LINUX, JAVA 등이 있다.**
- 해설:
  - JAVA는 프로그래밍 언어이며 운영체제가 아님.

17번 (2016년 2회-8번)

- 정답: ④ **실행 가능한 목적(object) 프로그램 생성**

- 해설:
    - 목적 프로그램 생성은 컴파일러와 링커의 역할임.

18번 (2016년 2회-12번)

- 정답: ③ **데이터의 압축 및 복원**
- 해설:
    - 데이터 압축 및 복원은 유틸리티 프로그램의 역할로 운영체제의 목적과 거리가 멂.

19번 (2016년 2회-13번)

- 정답: ① **서비스 프로그램**
- 해설:
    - 서비스 프로그램은 운영체제의 제어 프로그램에 포함되지 않음.

20번 (2015년 3회-12번)

- 정답: ① **Availability (사용 가능도)**
- 해설:
    - 시스템 자원을 요구할 때, 얼마나 신속하고 충분히 지원하는가를 나타내므로 사용 가능도가 정답임.

2주차

1열	프로세스, 스레드
2열	3/20
Status	완료

1. 프로세스(Process)

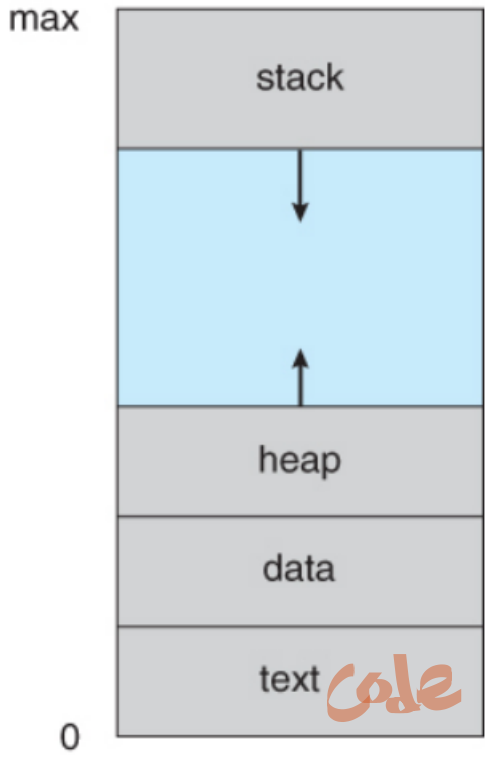
1. 프로세스 개요

- 프로세스 개념은 **1960년대 중반 Multics OS**에서 처음 사용되었으며, **IBM OS**에서는 *task*로 명명됨.
- **프로세스 정의**: *현재 실행 중이거나 곧 실행 가능한 PCB를 가진 프로그램.*
- 프로세스는 **프로세서에 의해 수행되는 프로그램 단위**로, 다음과 같은 특징을 가짐:
  - 실행 중인(Executing, Running) 프로그램
  - 비동기적(Asynchronous) 활동
  - 살아 있는(Live) 프로그램
  - 프로세스 제어 블록(PCB)을 가진 프로그램
  - 언제든지 실행 가능한(Executable) 프로그램

2. 프로그램과 프로세스의 차이

- **프로그램**: 디스크에 저장된 *수동적 실체(passive entity)*.
- **프로세스**: **프로그램 카운터(PC, Program Counter), 프로세스 제어 블록(PCB)**를 가지며, 실행 흐름을 제어하는 *능동적 실체(active entity)*.

3. 프로세스의 메모리 구조(중요) : 정적 영역(코드, 데이터), 동적 영역(스택, 힙)



1. **코드(Code) 영역, 텍스트(Text) 영역**: 실행 가능한 프로그램 코드 영역. CPU가 프로세스 실행 시 해당 영역 내용 참조
2. **데이터(Data) 영역**: 전역변수, 정적 변수, 작업 공간(work space), 사용자 버퍼 등. 프로그램이 실행될 때 생성되고, 종료될 때 반환함.
3. **힙(Heap) 영역** : 동적 메모리(malloc(), free()) 등에 사용되는 변수를 담는 영역. 크기 변동.
4. **스택(Stack) 영역**: 지역변수, 인자 리스트(argument list), 복귀 주소(return address), 스택 프레임 등.

---

2. 프로세스 제어 블록(PCB)

- *PCB(Process Control Block)**은 운영체제가 프로세스를 관리하기 위한 정보가 담긴 자료구조로, 다음과 같은 내용을 포함함.

2.1. PCB의 주요 정보

- **프로세스 상태**: 보류, 실행, 준비, 대기, 정지 등.
- **프로세스 ID**: 프로세스의 고유 식별 번호.
- **프로그램 카운터(PC)**: 다음 실행할 명령어의 주소.
- **레지스터 정보**: 누산기(Accumulator), 인덱스 레지스터, 스택 레지스터 등.
- **메모리 관리 정보**: 경계 레지스터(boundary register), 페이지 테이블(page table) 등.
- **계정(회계) 정보**: CPU 사용 시간, 계정번호, 작업번호 등.

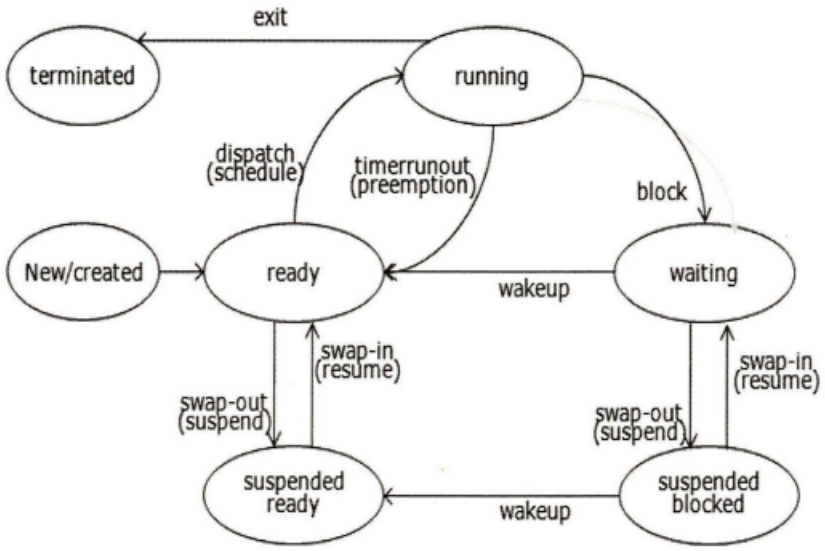
- **입출력 정보**: 입출력 요청 목록, 사용 중인 입출력 장치 및 개방된 파일 목록.

2.2. PCB의 역할

- **프로세스 상태 저장**: 실행 중이던 프로세스가 중단될 경우 현재 상태를 저장하고, 이후 재개 시 해당 정보를 활용.
- **문맥 교환(Context Switching)**: 운영체제가 프로세스를 전환할 때 PCB의 정보를 이용하여 중단된 프로세스를 복원.

---

3. 프로세스 상태와 상태 전이(중요)



3.1. 프로세스 상태

1. **생성 상태 (Created State)**
   - 사용자가 요청한 작업이 운영체제에 등록되어 PCB가 생성되는 상태.
   - 이후 **준비 상태(Ready State)** 또는 **지연 준비 상태(Suspended Ready State)** 로 전이될 수 있음.
2. **준비 상태 (Ready State)**
   - 프로세서(CPU) 할당을 기다리는 상태.
   - 모든 필요한 자원(메모리 등)을 할당 받았지만, 아직 실행되지 않음.
3. **실행 상태 (Running State)**
   - 프로세스가 CPU를 할당 받아 실행 중인 상태.
4. **대기 상태 (Waiting/Blocked State)**
   - 특정 자원(입출력 등)이 필요하여 대기 중인 상태.
   - 예: 입출력 요청 후 완료되기를 기다리는 경우.
5. **지연 준비 상태 (Suspended Ready State)**

- 준비 상태에 있던 프로세스가 메모리 부족으로 인해 디스크로 스왑됨.
- 메모리가 확보되면 다시 준비 상태로 전이됨.

6. **지연 대기 상태 (Suspended Waiting State)**
   - 대기 상태에서 메모리 부족으로 인해 스왑된 상태.
   - 요청한 자원이 할당되면 **지연 준비 상태**로 전이됨.

3.2. 프로세스 상태 전이

- **디스패치(dispatch)**: 준비 상태 → 실행 상태
  - 준비 상태의 프로세스 중 우선순위가 가장 높은 프로세스가 CPU를 할당 받아 실행 상태로 전환됨.
- **할당 시간 초과(time runout)**: 실행 상태 → 준비 상태
  - CPU 할당 시간이 종료되면 다시 준비 상태로 돌아감.
- **블록(block)**: 실행 상태 → 대기 상태
  - 실행 중인 프로세스가 입출력 요청 등으로 인해 CPU를 반환하고 대기 상태로 전환됨.
- **웨이크업(wake up)**: 대기 상태 → 준비 상태
  - 입출력 등의 요청이 완료되면 다시 준비 상태로 전환됨.

---

4. 프로세스와 스레드(매우 중요)

4.1. 프로세스와 스레드 비교

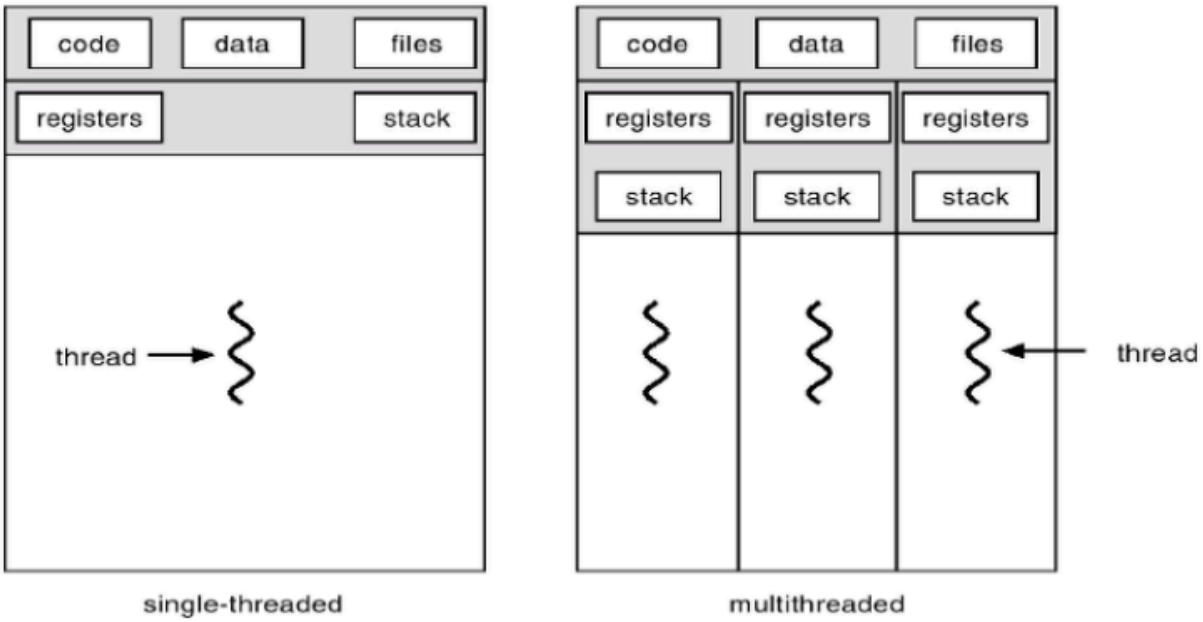
프로세스(Process)

- 실행 중인 프로그램을 의미하며, 독립적인 메모리 공간을 가짐
- 운영체제(OS)에 의해 관리되며, 서로 다른 프로세스 간에는 메모리를 공유하지 않음
- 하나 이상의 스레드를 포함할 수 있으며, 각각의 프로세스는 별도의 실행 흐름을 가짐

스레드(Thread)

- 프로세스 내에서 실행되는 최소 실행 단위
- 같은 프로세스 내에서 메모리와 자원을 공유하며, 다중 작업을 수행할 수 있음
- 하나의 프로세스에 여러 개의 스레드를 두어 병렬 처리를 효율적으로 수행 가능

4.2. 싱글 스레드와 멀티 스레드



싱글 스레드 (Single Thread)

- 하나의 스레드만을 사용하여 작업을 처리
- 순차적으로 실행되므로 하나의 작업이 끝나야 다음 작업을 수행할 수 있음
- 구현이 간단하지만, 대기 시간이 길어질 수 있음

멀티 스레드 (Multi Thread)

- 하나의 프로세스 내에서 여러 개의 스레드가 동시에 실행됨
- 병렬 처리를 통해 응답성을 향상시키고 CPU 사용률을 최적화할 수 있음
- 스레드 간 자원 공유가 가능하지만 동기화 문제 발생 가능

4.3. 커널 스레드와 사용자 스레드(중간고사 범위 X)

커널 스레드 (Kernel Thread)

- 운영체제 커널이 직접 관리하는 스레드
- 시스템 호출을 통해 커널에서 스레드를 생성하고 관리
- 커널에서 직접 스케줄링을 담당하므로 안정성이 높지만, 스레드 간 전환 비용이 발생

사용자 스레드 (User Thread)

- 사용자 공간에서 관리되는 스레드
- 커널에 직접적인 개입 없이 사용자 수준에서 생성 및 관리 가능

- 속도가 빠르고 효율적이지만, 커널에서 직접 제어하지 않기 때문에 하나의 스레드가 블록되면 전체 프로세스가 영향을 받을 수 있음

3주차

1열	프로세스 스케줄링
2열	3/27
Status	시작 전

1. 스케줄링 개요

1. **정의**
   - 프로세스에게 CPU와 처리기를 효율적으로 할당하기 위한 운영체제의 정책
2. **목표**
   - CPU 이용률 및 처리율 **최대화**
   - 반환시간, 대기시간, 응답시간 **최소화**
3. **용어**
   - **CPU 이용률(CPU Utilization)**: CPU가 바쁘게 동작한 비율
   - **처리율(Throughput)**: 단위 시간 내 완료된 프로세스 수
   - **반환시간(Turnaround Time)**: 도착부터 종료까지 걸린 시간
   - **대기시간(Waiting Time)**: 준비 큐에서 대기한 시간
   - **응답시간(Response Time)**: 요청 후 첫 응답까지의 시간

---

2. 스케줄링의 종류

2.1. 기능별 분류

1. **장기 스케줄링 (Long-Term Scheduling)**
   - 어떤 작업을 시스템에 받아들일지 결정 (작업 큐 → 준비 큐)
2. **중기 스케줄링 (Medium-Term Scheduling)**
   - 메모리 과부하 방지를 위해 일부 프로세스를 임시 중단
3. **단기 스케줄링 (Short-Term Scheduling)**
   - 어떤 준비 프로세스에게 CPU를 줄지 결정 (실시간)

2.2. 방법별 분류

1. **선점형(Preemptive)**: 현재 실행 중인 프로세스를 중단 가능
2. **비선점형(Non-preemptive)**: 현재 프로세스가 종료될 때까지 기다림

---

3. CPU 스케줄링 알고리즘

3.1. 선입선출 스케줄링 (FCFS)

- **유형**: 비선점형
- **개념**: 먼저 도착한 프로세스부터 순서대로 실행
- **특징**:
  - 간단하고 공정함
  - **호위 효과, 호송 효과(Convoy Effect)** 발생 가능
  - 대화식/시분할 시스템에 부적합

3.2. 최단 작업 우선 스케줄링 (SJF)

- **유형**: 비선점형
- **개념**: 실행 시간이 가장 짧은 작업을 먼저 실행
- **특징**:
  - 평균 대기 시간 최소화
  - 실행 시간 예측이 어려움
  - 긴 작업은 **기아(starvation)** 발생 가능
  - **에이징(Aging)** 기법으로 해결 가능

3.3. 최단 잔여 시간 우선 스케줄링 (SRT)

- **유형**: 선점형
- **개념**: 남은 실행 시간이 가장 짧은 프로세스에게 CPU를 할당
- **특징**:
  - 새로 도착한 프로세스가 더 짧으면 현재 작업 선점
  - 선점 오버헤드 존재
  - 기아 발생 가능

3.4. HRN 스케줄링 (Highest Response Ratio Next)

- **유형**: 비선점형
- **개념**: 응답 비율이 높은 프로세스를 먼저 실행
  - 응답 비율 = (대기시간 + 서비스시간) / 서비스시간
- **특징**:
  - SJF의 기아 문제 해결
  - 짧은 작업과 오래 기다린 작업에 유리

3.5. 우선순위 스케줄링 (Priority Scheduling)

- **유형**: 선점형/비선점형 모두 가능
- **개념**: 우선순위가 높은 프로세스에게 CPU를 할당
- **특징**:
  - 낮은 우선순위는 **기아** 발생 가능 → 에이징으로 해결
  - **정적/동적 우선순위 방식** 있음
    - 정적: 단순하지만 비적응적
    - 동적: 적응성 높지만 오버헤드 큼

3.6. 라운드로빈 스케줄링 (Round-Robin, RR)

- **유형**: 선점형
- **개념**: 각 프로세스에 일정한 시간 할당량(Time Quantum)을 주고 순환 실행
- **특징**:
  - 대화식/시분할 시스템에 적합
  - 시간 할당량이 적절해야 효율적
  - 문맥 교환 비용 발생

3.7. 다단계 큐 스케줄링 (Multi-Level Queue, MLQ)

- **유형**: 선점형/비선점형
- **개념**: 작업 특성에 따라 여러 큐로 나누고, 큐별로 다른 스케줄링 적용
- **특징**:
  - 큐 간 이동 불가
  - 고정된 큐 구조
  - 우선순위 큐 기반

3.8. 다단계 피드백 큐 스케줄링 (Multi-Level Feedback Queue, MFQ)

- **유형**: 선점형
- **개념**: 프로세스 특성 및 상태에 따라 큐 간 이동 가능
- **특징**:
  - **짧은 작업, I/O 중심 작업에 유리**
  - 동적 우선순위 및 에이징 기법 사용
  - 높은 적응성과 유연성

- 시간 할당량은 큐마다 다름
- **단점**: 구현 복잡, 시스템 부하 증가, 기아 발생 가능

📌 참고: 기아(Starvation)와 에이징(Aging)

- **기아(Starvation)**: 우선순위가 낮아 계속 CPU를 할당받지 못하는 상태
- **에이징(Aging)**: 오래 대기한 프로세스의 우선순위를 점진적으로 높여 기아 해결

4주차

1열	프로세스 스케줄링 연습
Status	시작 전

참고 : 해설 및 답을 푼 대로 작성하였으나.... 계산 이슈로 인해 답이 틀릴 수도 있습니다. 본인 답이 맞는 거 같다면 강경아T & 박건우 T에게 찾아와서 정정 부탁드립니다....(소정의 상품 있음)

1. 프로세스 스케줄링 연습(3주차)

▼ 문제 1 : FCFS, SJF, SRT, HRN

- 다음 5개의 프로세스가 있다고 하자. FCFS, SJF, SRT, HRN 스케줄링 알고리즘을 각각 적용하였을 때, **평균반환시간**과 **평균대기시간**을 구하시오.

프로세스	도착 시간	버스트 시간
P1	0	3
P2	1	7
P3	3	2
P4	5	5
P5	6	3

▼ 해설1 : FCFS

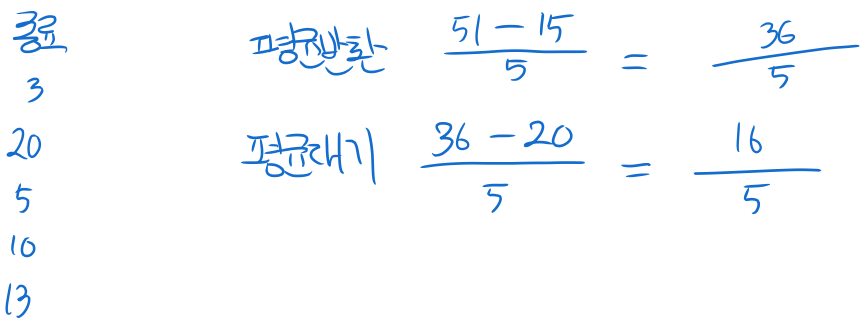
종료
3
10
12
17
20

평균반환 = $\frac{62 - 15}{5} = \frac{47}{5}$ = 종료시간의 합 - 도착시간의 합 의 평균

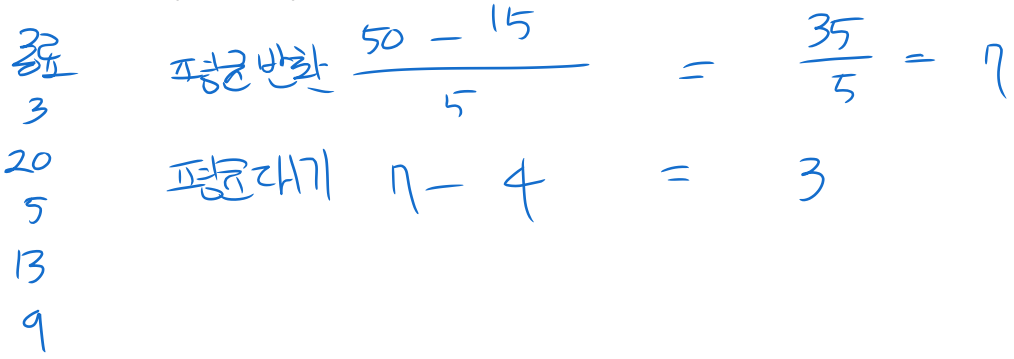
평균대기 = $\frac{47 - 20}{5} = \frac{27}{5}$

▼ 해설2 : SJF

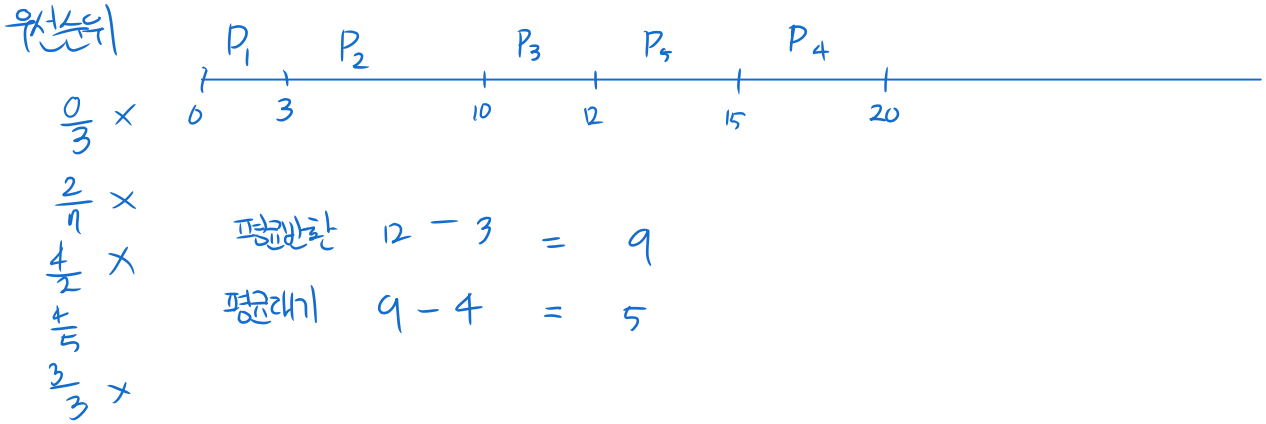
- SJF 적용 시, 아래 표로 정리할 수 있음



▼ 해설3 : SRT(선점형 SJF)



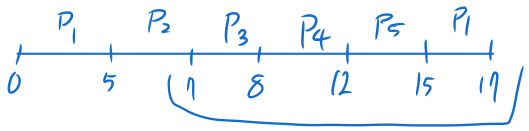
▼ 해설4 : HRN(이해 잘 하기)



▼ 문제 2 : 라운드 로빈, 우선순위

- 다음 5개의 프로세스가 있다고 하자. 해당 프로세스는 단일 처리기 시스템의 대기 큐에 p1~p5의 순서로 들어가 있다. 또한, 우선순위 값이 작을수록 우선순위가 높다고 하자.

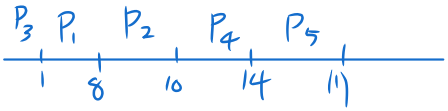
프로세스	우선순위	CPU 버스트 시간
P1	2	7



P2	3	2
P3	1	1
P4	3	4
P5	4	3

- 라운드 로빈 스케줄링 알고리즘을 적용할 경우, 프로세스 p1~p5의 평균 대기 시간을 구하시오. (단, time slice는 5이다.)
- 비선점 우선순위 스케줄링 알고리즘을 적용할 경우, 프로세스 p1~p5의 평균 대기 시간을 구하시오. (단, 우선순위가 같을 경우 FCFS 정책에 따른다.)
- 라운드 로빈 스케줄링 알고리즘을 적용할 때, CPU 할당량을 무한대로 설정할 경우, 프로세스 p1~p5의 평균 대기 시간을 구하시오.

▼ 정답 2



▼ 해설 2

▼ 문제 3 : MFQ → 4주차 진행 예정, 도전하기

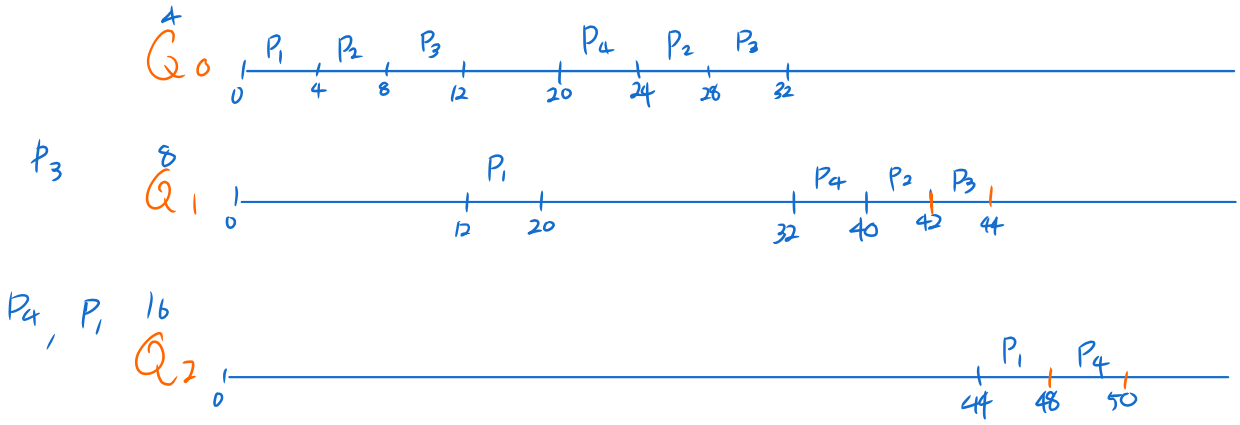
- 다단계 피드백 큐를 이용하여 CPU 스케줄링을 하고자 한다. 물음에 답하시오.

- 3단계 큐를 사용하며, CPU 사용 권한이 높은 단계부터 낮은 단계로 큐0, 큐1, 큐2라 하고 모든 프로세스는 생성 후 큐 0에 할당된다.
- 큐 0, 큐 1, 큐 2에 저장된 프로세스는 들어오는 순서대로(FCFS) 각각 4초, 8초, 16초의 CPU 할당을 가지며, 한 번 CPU 할당을 배정받으면 할당 시간을 모두 소모하거나, 종료될 때 까지 다른 프로세스에게 CPU 사용권을 내어주지 않는다.
- 큐 1과 큐 2에 놓인 프로세스가 각 대기시간 15초, 30초에 처리되지 않으면 해당 프로세스는 한 단계 높은 큐로 이동한다.
- 4개의 프로세스의 도착시간과 CPU 요구시간은 다음과 같다.

프로세스	도착 시간	CPU 버스트 시간
P1	0	16
P2	4	10
P3	8	10
P4	15	14

- 큐 1에 저장된 P1의 작업처리 시작 시간은?
- 큐 0에 있는 P4가 큐 1로 이동하는 시간은?
- 시점 21초에서 큐 1에 저장된 프로세스는?
- 네 개의 프로세스들 중 할당된 작업이 가장 늦게 완료되는 프로세스는?

▼ 정답 3



cpu 스케줄링 연습문제(심화).pdf

cpu 스케줄링 추가.pdf

심화문제

▼ 문제 1

1번 : P2

2번 : P9

해설 - 1 : t1 시점에는 P5가 실행중이고, 5ms동안 실행되므로(잔여 시간), t1 + 5ms 시점에는 종료된다.

따라서, t1 + 5ms 시점에, FCFS를 적용 시, Ready Queue에서 생성 시각이 가장 빠른 P2가 선택되고, 이는 t1 + 55ms까지 지속되므로

t1 + 10ms 시점에, 실행되는 프로세스는 P2 이다.

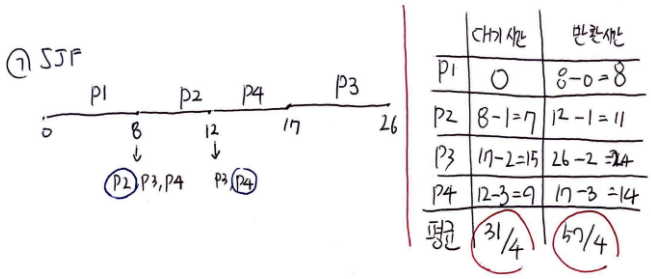
해설 -2 : 위와 동일하게, t1 + 5ms 까지는 P5가 실행되다가, t1 + 5ms 에서 CPU 버스트가 가장 작은 P7이 실행되어 t1 + 20ms에 종료된다. 중요한 것은, 이 시점 이전인 t1 +15ms에서 P9가 입출력 버스트를 완료하고 Ready Queue에 진입하여, t1 + 20ms 시점에서 잔여 실행 시간이 가장 작은 프로세스는 P3가 아니라 P9이 된다.

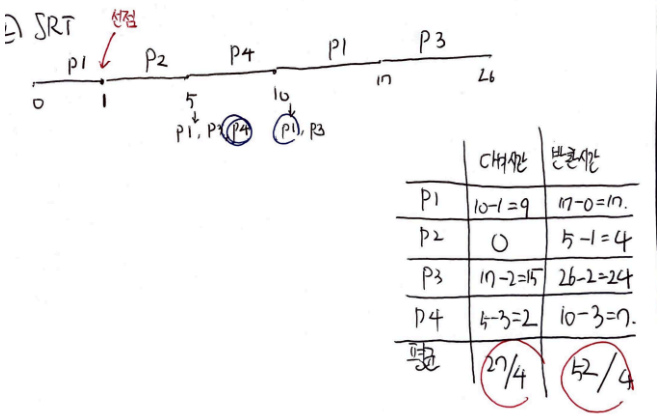
▼ 문제 2

ㄱ의 평균대기 : 31/4, ㄱ의 평균반환 : 57/4

ㄴ의 평균대기 : 26/4, ㄴ의 평균반환 : 52/4

▼ 해설 : 스스로 풀고 답 보십시오.





▼ 문제 3 : 참고, 라운드 로빈이 도착시간이 동일할 경우 프로세스 이름 순서로!

1번 : 2회, 71/5

선점 : 2초에 p1→p2

선점 : 6초에 p2→p5

p1 : 42 - 2

p2 : 15 - 6

p3 : 25 - 6

p4 : 9 - 6

p5 : 0

2번 : 7회

Gantt Chart 구성 (간략 순서)

1. **0~6**: P1 (남은 14)
2. **6~12**: P2 (남은 8)
3. **12~18**: P3 (남은 11)
4. **18~24**: P4 (남은 0) ✅ 완료
5. **24~30**: P5 (남은 0) ✅ 완료
6. **30~36**: P1 (남은 8)
7. **36~42**: P2 (남은 2)
8. **42~48**: P3 (남은 5)

9. **48~54**: P1 (남은 2)
10. **54~56**: P2 (2초 → 완료)
11. **56~61**: P3 (5초 → 완료)
12. **61~63**: P1 (2초 → 완료)

✅ 선점 횟수 계산 (TQ = 6)

- P1: 3회 교체 (6, 30, 48 → 3번 선점)
- P2: 2회 교체 (12, 36 → 2번 선점)
- P3: 2회 교체 (18, 42 → 2번 선점)
- **총 선점 횟수 = 3(P1) + 2(P2) + 2(P3) = 7회**

3번 : 55회

총 선점 횟수 = 19 + 13 + 16 + 5 + 2 =
55회

중요 : 이렇게, time slice가 작아지게 되면 cpu 효율이 아주 나빠지게 됨

▼ 문제 4

1번 해설 : 먼저 P0이 수행되고, 80초 동안 반환하지 않는다.

80초 시점에는, p1 p2 p3가 진입하였는데 우선순위가 가장 높은 p1 p3 중에서 하나가 실행된다.

p1 p3 중에서 먼저 도착한 p1이 실행되고, 105초까지 선점되지 않고 수행된다.

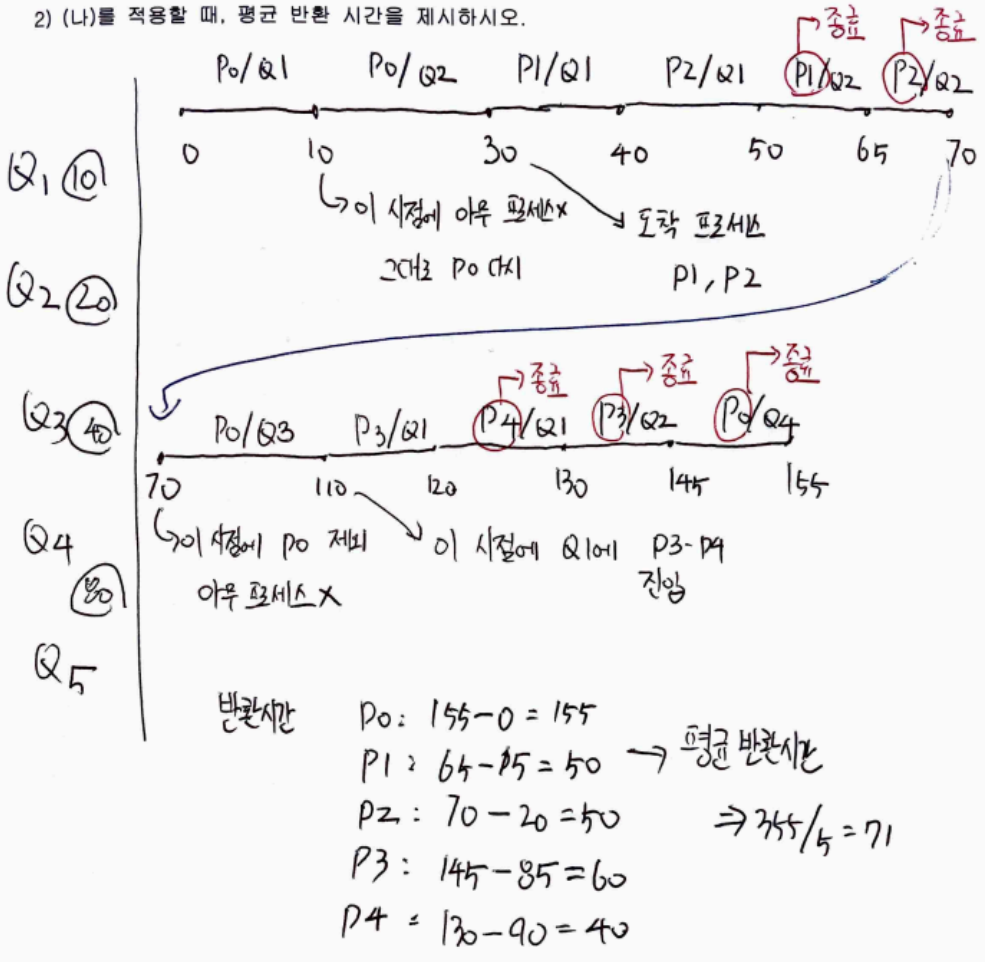
105초 시점에서는 p2, p3, p4가 남는데 우선순위가 p4 p3 p2 이므로, p4 p3 p2 순으로 실행된다.

따라서 순서는 p0 p1 p4 p3 p2 이다.

2번 해설 : 손 풀이로 대체합니다.

▼ 스스로 풀고 답 보십시오.

2) (나)를 적용할 때, 평균 반환 시간을 제시하시오.



반환시간

P0: 155 − 0 = 155

P1: 64 − 15 = 50 → 평균 반환시간

P2: 70 − 20 = 50 ⇒ 355/5 = 71

P3: 145 − 85 = 60

P4: 130 − 90 = 40

5주차

1열	임계영역 문제, SW적 해결법
Status	시작 전

1. 병행 프로세스

▼ 개관

- 병행 프로세스 : 동시에 실행되는 두 개 이상의 프로세스를 말함.

▼ 예시(Java)

```
public class Main {
    static int count = 0;

    public static void main(String[] args) throws InterruptedException {
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 10000; i++) {
                count++;
            }
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 10000; i++) {
                count--;
            }
        });

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        System.out.println("result: " + count);
    }
}
```

- 의도한 대로라면 0이 나와야 하나, 그렇지 않음.
- 이유 : 더하기 빼기 연산을 덮어쓰는 일 → 공유 변수를 덮어 쓰는 일이 발생하여 의도한 대로 동작하지 않음. (결정성이 없다, 또는 race condition)
- 이처럼, 여러 프로세스가 병행되어 동작할 때 생기는 문제를 해결하는 방법을 학습할 예정

2. 임계영역 문제

▼ 소개

각 프로세스는 임계 영역이라는 코드 부분을 포함하고 있고, 이 안에서 다른 프로세스와 공유하는 변수를 변경하거나 테이블을 변경하거나 파일을 쓰거나 하는 등의 작업을 수행한다. 이 때 중요한 사실은 한 프로세스가 자신의 임계 영역에서 작업을 수행하는 동안에는 다른 프로세스가 이 영역에 들어갈 수 없다는 것이다.

▼ 충족 조건

임계 영역 문제를 해결하기 위한 해결안은 **다음 세 가지 조건**을 충족해야 한다.

1. 상호 배제(Mutual exclusion) : 한 프로세스가 **자기 임계 영역에서 실행된다면 다른 프로세스들은 그 임계 영역에서 실행될 수 없다.**
2. 진행(Progress) : **임계 영역에서 실행 중인 프로세스가 없고 진입하려는 프로세스들이 있다면 이들 중에 진입시킬 프로세스를 선택하여 진입**시켜야 한다. 이 선택은 무기한으로 연기되어서는 안 된다.
3. 한정 대기(Bounded waiting) : 프로세스가 임계 영역에 진입하려는 요청을 한 뒤로 요청이 허용될 때까지 **다른 프로세스들이 임계 영역에 진입하도록 허용하는 횟수에는 제한이 있어야 한다.**

▼ 프로세스 구조

```
while (1){

// entry section : 임계 영역으로의 진입 허가를 요청하는 코드

// critical section

// exit section : 임계 영역에서 작업을 마친 후 마무리 하는 코드

// remainder section : 프로그램의 나머지 코드 영역

}
```

▼ 임계영역 알고리즘

- 동기화 하드웨어 : 하드웨어에서 원자적 연산 명령어만 제공 → 복잡한 제어 없이 가능
- SW적 해결기법 : 하드웨어 도움 없이, 논리적인 절차를 통해 해결
- 고수준 동기화 도구 : 세마포어, 모니터

3. 2개 프로세스 간 상호 배제

▼ 1st algorithm : 전역 변수 turn 활용(0과 1로만 세팅)

- turn의 의미 : 누가 임계 영역에 들어갈 차례인지 나타내는 변수
- turn이 0 → P0이 들어갈 차례, 1 → P1이 들어갈 차례

```
// P0 구조
while (1){
 while(turn != 0);

 // critical section

 turn = 1; // exit section

 // remainder section
}
```

```
// P1 구조
while (1){
 while(turn != 1);

 // critical section

 turn = 0; // exit section

 // remainder section
}
```

- 조건 검증
- 상호 배제 : 보장(turn 값에 따라 진입 여부 결정)
- **진행 : 보장할 수 없음(turn 값이 0인데, P0이 잔류 구역 작업중, P1만 진입 희망할 경우)**
- 한정 대기 : 보장(P0이 실행중이고, P1이 대기중일 때, P0이 끝나면 P1 진입 가능)
- 문제점 : turn 변수만으로는 해당 프로세스가 critical section에 있는 지, remainder section에 있는 지 구분 안됨.

▼ 2nd algorithm : flag 배열 활용, 각 배열 원소의 값은 0과 1로만 세팅

- flag 배열의 의미 : 각 프로세스가 임계영역에 들어가고 싶은지/아닌지를 나타내는 배열
- flag[0] 의 의미 : P0이 임계구역에 진입하기를 희망함. P0이 1이면 임계구역에 진입하기를 희망

```
// P0의 구조
while (1){
 flag[0] = 1;
 while(flag[1]);

 // critical section

 flag[0] = 0;

 // remainder section
 }
```

```
// P1의 구조
while (1){
 flag[1] = 1;
 while(flag[0]);

 // critical section

 flag[1] = 0;

 // remainder section
 }
```

- 조건 검증
- 상호 배제 : 보장(상대의 flag 값이 0이면 혼자 진입 가능)
- **진행 : 보장할 수 없음(동시에 flag 값이 1이면, 무한 루프)**

▼ 3rd algorithm : flag 배열 활용, 각 배열 원소의 값은 0과 1로만 세팅

```
// P0의 구조
while (1){
 while(flag[1]);
  flag[0] = 1;

 // critical section

 flag[0] = 0;

 // remainder section
 }
```

```
// P1의 구조
while (1){
 while(flag[0]);
  flag[1] = 1;

 // critical section

 flag[1] = 0;

 // remainder section
 }
```

- 조건 검증
- **상호 배제 : 보장할 수 없음(동시에 진입)**

▼ 4th algorithm(Peterson algorithm) : flag 배열, turn 변수 활용

```
// P0의 구조
while (1){
 flag[0] = 1;
 turn = 1;
 while(flag[1] && turn == 1);

 // critical section

 flag[0] = 0;

 // remainder section
 }
```

```
// P1의 구조
while (1){
 flag[1] = 1;
 turn = 0;
 while(flag[0] && turn == 0);

 // critical section

 flag[1] = 0;

 // remainder section
 }
```

- 검증
- 상호 배제 : 보장할 수 있음(동시에 진입할 경우, turn이 0 또는 1 중에 하나로 결정되기 때문)
- 진행 : 보장할 수 있음(P0이 요청 + P1이 임계 영역 수행중 → P1이 끝나고 flag 값 변경 → P0 진입 가능)
- 한계 대기 : 보장할 수 있음(P0 요청 + P1 임계 영역 수행 중 → P1이 임계 영역 끝나면 바로 P0 진입 가능)

▼ Peterson algorithm의 문제점

- 하드웨어 수준의 보장 어려움 : flag, turn을 동시에 읽기 쓰기를 전제하지만, 이는 어려움
- 현대 CPU에 부적합 : 현대 CPU는 멀티 코어이지만, 이는 싱글 코어를 전제로 함
- 확장 부족 : 2개가 아닌 n개의 프로세스에 대해서는...?

4. N개 프로세스 간 상호 배제

▼ Lamport’s Bakery Algorithm(중요)

- 변수 설명 : choosing 배열 : 번호표를 아직 뽑는중인지 확인하는 배열 → 값이 0이면 아직 뽑지 않았거나, 뽑았거나. 값이 1이면 번호표를 뽑는 중
- 변수 설명 : number 배열 : 번호표의 값. 이 값이 작을수록 임계영역 진입 우선순위가 높음
- 연산 설명 : (number[j], j) < (number[i], i) → number[j]와 number[i] 비교. 두 값이 같다면 j와 i 비교

```
//프로세스 i의 구조, n개의 프로세스가 있다 가정

// choosing[], number[]은 초기 값이 모두 0

while(1) {

   ...
   choosing[i] = 1;
   number[i] = max(number[0], ..... number[n-1]) + 1 ;
   choosing[i] = 0;

   for(j=0; j<n; j++) {
      while(choosing[j]);
      while(number[j] && (number[j], j) < (number[i], i));
   }
   // Critical Section
   ...
   number[i] = 0;
   ...

   // Remainder Section

   }
```

▼ 알고리즘 수행 순서

1. 진입한 프로세스는 자신의 choosing을 1로 세팅(이제 번호표 뽑습니다!)
2. 현재까지의 번호표 값보다 더 큰 값을 받기로 함(마지막에 들어와서)
3. 번호표를 뽑은 후, choosing 값을 0으로 세팅(번호표 뽑았습니다!)
4. 0번부터 n-1번까지 프로세스 확인
5. 혹시 번호표 안 뽑은사람이 있는가 확인
6. 모두 번호표 뽑았으면, 번호표 값 & 프로세스 번호 보면서 확인

7. 만약 번호표 값이 0이다 → ~~번호표를 안뽑았다(아마 걸려저서 아님)~~, 이미 임계영역 쓴 사람이다
8. 자기 값이 제일 작다고 판단(for 루프 모두 확인 결과) → 진입
9. 다 쓰고 number 0으로 하여 종료

▼ 보장 부분

```
do{

        choosing[i] = true;      // 번호표를 뽑지 못한 상태
        number[i] = max(number[0],number[1], ..... , number[n-1]) + 1;  -------- ①
        choosing[i] = false;
        for( j=0; j<n; j++){                                              -------- ②
                while(choosing[j]);
                while( (number[j]!=0) && ((number[j], j) < (number[i], i)) );
        }
        critical section
        number[i] = 0;                                                    --------- ③
        remainder section
}while(1);
```

① 한계대기 만족 부분 (FIFO) ② 상호배제(조건) 만족 부분 ③ 진행 만족 부분

5. 챌린지

▼ race condition

- 다음 프로세스 A, B는 공유 변수 count를 공유하며 병행 실행되고 있다.
- 공유 변수 count의 초깃값은 0이며, a와 b는 각 프로세스에서만 사용되는 변수이다. 아래는 프로세스 별 의사 코드이다.

```
// Process A
// a는 지역 변수
for(4회 반복){
a = count;
a = a * 3;
count = a;
}
```

```
// Process B
// b는 지역 변수
for(4회 반복){
b = count;
b = b + 1;
count = b;
}
```

- count가 가질 수 있는 최댓값, 최솟값은?
- 최솟값 : 0
- 최댓값 : 324(4 * 3 * 3 * 3 * 3)