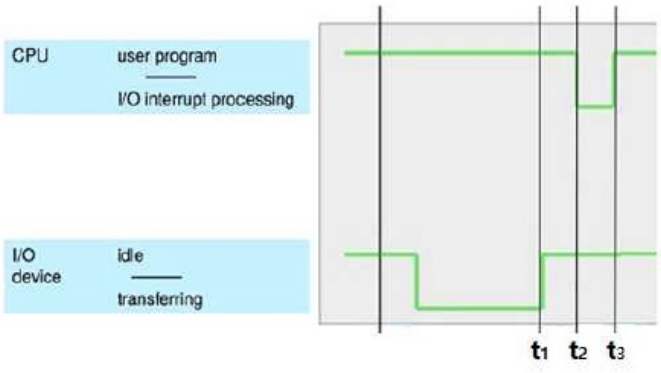
Q1.

CPU와 I/O device간 timeline그림을 통해 예측할 수 있는 작업 상황에 대해 설명하고, t1, t2, t3에서 CPU와 I/O device 사이에 발생하는 이벤트 작업에 대해 매 시간별로 설명하시오.



- 인터럽트 기반 입출력 작업. I/O device가 데이터를 전송하는 동안 CPU는 user program을 실행하고, 전송이 끝나면 인터럽트 요청 신호를 보내고 CPU가 I/O device의 작업을 확인함.
- t1: I/O device에서 데이터 전송이 끝나고 대기상태로 접어듦. CPU에 I/O 인터럽트 요청 신호를 보냄
- t2: CPU에서 작업을 백업하고 인터럽스 서비스 루틴을 실행함
- t3: CPU가 인터럽스 서비스 루틴을 종료하고 다시 user program 으로 돌아가서 진행하던 프로그램을 이어서 진행함

Q2.

마이크로커널(microkernel) 구조와 모놀리틱커널(monolithic kernel) 구조의 차이점에 대해서 커널 서브시스템(kernel subsystem)의 프로그램 실행 레벨과 보호영역 (protection domain) 관점에서 서술하시오.

- 마이크로 커널: 시스템의 안정성과 격리를 강화하기 위해 보호된 환경을 제공
- 모놀리틱 커널: 커널 내의 모든 서비스가 동일한 실행 레벨과 보호 영역 내에서 동작

1. 프로그램 실행 레벨

- 마이크로 커널: 최소한의 핵심 기능만을 커널 내에서 직접 실행하고 나머지 서비스는 사용자 모드에서 실행. 사용자 공간과 커널 공간 간의 분리가 이루어져 프로그램 실행 레벨이 높아짐
- 모놀리틱 커널: 모든 서비스 및 드라이버가 커널의 단일 실행 레벨에서 동작. 모든 커널 서비스가 커널의 주소 공간 내에서 실행되는 것을 의미하며, 커널 내에서 공유된 실행 환경을 가짐

1. 보호 영역

- 모놀리틱 커널: 단일 주소 공간 내에서 모든 서비스를 실행하므로, 커널 내의 서비스는 보호 영역의 분리 없이 서로 직접 접근 가능. 커널 내의 오류나 버그로 인해 전체 시스템의 안정성에 영향을 미칠 수 있음
- 마이크로 커널: 커널 서비스가 서로 다른 보호 영역에서 실행됨. 커널은 사용자 모드와 커널 모드 간의 보호를 제공하여 사용자 모드에서 실행되는 서비스가 커널의 내부에 직접 접근하지 못하도록 함. 커널 내의 오류가 전체 시스템에 미치는 영향을 최소화

Q3.

DMA(direct memory access)를 사용하여 CPU의 실행 부하(execution load)없이 고속 입출력 장치들을 사용하고자 한다. 이때 장치로의 메모리 연산이 완료되었음을 CPU가 알 수 있는 방법이 무엇이며, 그 방법과 트랩(trap)과의 차이에 대해서 서술 하시오.

인터럽트: DMA 컨트롤러는 데이터 전송이 완료되면 CPU에게 인터럽트를 발생시킴. 인터럽트는 CPU에게 데이터 전송이 완료되었음을 알리고, CPU는 인터럽트 서비스 루틴을 실행

특성	인터럽트	트랩
발생 원인	외부 이벤트 또는 하드웨어에서 발생	프로그램 내에서 명시적으로 설명하거나 예외 상황 발생
발생 시점	비동기적 (비정상적)	동기적 (정상적 또는 예외적)

특성	인터럽트	트랩
주요 용도	입출력 완료, 하드웨어 예외 등	예외 처리, 소프트웨어 이벤트 처리
작업 처리	현재 실행중인 작업을 중단하고 인터럽트 서비스 루틴을 수행	현재 프로그램 상태를 일시 중단하고 트랩 핸들러 루틴을 수행

Q4.

임계구역 문제(Critical-section problem)의 해결책은 세 가지 요구사항인 Mutual exclusion, Progress, 그리고 Bounded waiting을 만족시켜야 하는데, 이러한 세 가지 요구사항에 대해 설명하시오.

- Mutual exclusion (상호 배제)
  임계구역에 오직 하나의 프로세스만이 들어갈 수 있도록 보장해야함. 한 프로세스가 임계구역에서 실행중일 때 다른 프로세스는 들어가지 못하도록 해야함. 데이터 일관성 유지에 도움
- Progress (진행)
  임계구역에 들어가려는 프로세스가 없을 때, 새로운 프로세스가 들어가는 것을 보장해야함. 모든 프로세스가 임계구역에 접근하려고 할 때 어느 한 프로세스가 무한히 대기하지 않아야 함. 교착상태 (deadlock)을 방지하고 시스템이 효율적으로 동작하도록 함
- Bounded waiting (한정된 대기)
  한 프로세스가 다른 프로세스를 대기시키는 시간을 제한함. 특정 프로세스가 너무 오랫동안 다른 프로세스에게 대기시키지 않아야 함. 프로세스가 임계구역에 들어가기 전에 일정한 시간 또는 횟수의 대기를 보장할 수 있음

Q5.

Multiprocessor 환경에서 발생될 수 있는 문제 중 하나인 cache coherency에 대해 설명하시오.

- Cache coherency (캐시 일관성): 다중프로세서 시스템이나 멀티코어 프로세서 시스템에서 발생할 수 있음. 각 프로세서 또는 코어가 자체 캐시를 가지고 있고, 메모리의 데이터를 캐시에 복사하는 동안 발생할 수 있음.
- 캐시 일관성 문제가 발생하면 메모리의 데이터와 캐시에 있는 데이터 간에 불일치가 발생할 수 있으며, 이로 인해 동작이나 데이터 오류가 발생할 수 있음.
- 캐시 일관성 프로토콜을 통해 데이터 일관성을 유지하고 오류를 방지할 수 있음

Q6.

CPU 스케줄링 알고리즘들(scheduling algorithm)인 FIFO(First In First Out), SJF(Shortest Job First), RR(Round Robin)의 스케줄링 방식에 대해 설명하고, 장단점 을 비교 분석하시오.

알고리즘	스케줄링	장점	단점
FIFO	먼저 들어온 순서대로 CPU 할당	구현이 간단, 공정한 스케줄링	평균 대기 시간이 길어질 수 있음, 기아 (Starvation) 현상 발생 가능
SJF	실행 시간이 가장 짧은 작업 선호	평균 대기 시간을 최소화, CPU 이용률을 향상시킴	실행 시간 예측이 어려울 때 사용할 수 없음, 짧은 작업은 빠르게 실행되지만, 긴 작업은 대기할 수 있음
RR	시간 할당량 동안 CPU 할당	공정한 스케줄링, 응답 시간 짧아짐	시간 할당량 설정이 어렵고, 설정에 따라 성능이 달라질 수 있음

기아 (Starvation) 현상: 짧은 작업이 긴 작업을 기다리는 현상

Q7.

선점형 스케줄러(preemptive scheduler)와 비선점형 스케줄러(non-preemptive scheduler)를 설명하고, 응답성 및 예측 가능성 측면에서 비교 분석하시오.

- 선점형 스케줄러
  - 현재 실행중인 프로세스를 중단하고 다른 프로세스로 CPU를 할당할 수 있는 스케줄링 방식
  - 프로세스가 CPU를 점유하고 있는 동안 다른 높은 우선순위의 프로세스가 도착하면, 실행을 중단하고 높은 우선순위의 프로세스에게 CPU 할당
  - 예시: 우선순위 기반 스케줄러, 라운드 로빈 스케줄러
  - 응답성: 프로세스의 중단 및 전환을 허용하기 때문에 응답성 높음
  - 예측가능성: 우선순위와 시간 할당량과 같은 매개변수를 사용하여 스케줄링을 조절할 수 있어서 예측 가능성이 높음
- 비선점형 스케줄러
  - 현재 실행중인 프로세스가 종료하거나 대기상태로 들어가야만 CPU를 다른 프로세스에 할당하는 방식
  - 프로세스가 CPU를 점유하고 있는 동안에는 다른 높은 우선순위의 프로세스가 도착하더라도 대기해야함
  - 예시: SJF (Shortest Job First) 스케줄러
    응답성: 현재 실행중인 프로세스를 마칠 때까지 기다려야 하므로 응답성이 낮음
  - 예측가능성: 현재 실행중인 프로세스를 기다려야 하기 때문에 예측 가능성이 낮음

Q8.

Short-term scheduler와 Long-term scheduler의 차이점을 설명하고, (1)과 (2) 상황 이 발생하는 경우, 스케줄러(scheduler)와 큐(queue) 관점에서 자세히 설명하시오.
(1) If all processes in the ready queue are I/O bound
(2) If all processes in the ready queue are CPU bound

- Short-term scheduler (CPU 스케줄러):
  - 시점**:** 실행 중인 프로세스가 I/O 요청 또는 완료되었을 때, 혹은 타이머 인터럽트와 같은 이벤트가 발생할 때 실행됩니다. 즉, CPU를 어떤 프로세스에게 할당할지를 짧은 시간 간격으로 결정합니다.
  - 목적**:** 주로 CPU 자원을 효율적으로 활용하여 프로세스 스케줄링을 수행합니다. 현재 실행 중인 프로세스를 중단하고, 다른 프로세스에게 CPU를 할당합니다.
- Long-term scheduler (Job 스케줄러):
  - 시점**:** 시스템에 새로운 프로세스가 생성되거나, 프로세스가 종료되었을 때 실행됩니다. 장기적인 시스템 자원 관리를 위해 주기적으로 실행됩니다.
  - 목적**:** 프로세스를 메모리에 로드하거나 메모리에서 제거하여, 시스템 자원을 효율적으로 관리하고 CPU 스케줄러에게 실행할 프로세스를 선택하는 후보 집합(ready queue)을 관리합니다.

(1) If all processes in the ready queue are I/O bound:
- 스케줄러 관점**:** Short-term scheduler는 I/O 바운드 프로세스 중 어떤 프로세스를 실행할 것인지 선택해야 합니다. I/O 바운드 프로세스는 CPU 실행 시간 대비 I/O 대기 시간이 길기 때문에, I/O 대기 시간이 적은 프로세스를 선택하여 CPU를 효율적으로 활용할 수 있습니다.
- 큐 관점**:** Ready queue에는 모두 I/O 바운드 프로세스가 있으므로, Long-term scheduler가 이러한 프로세스들을 메모리에 로드한 후, Short-term scheduler가 실행할 프로세스를 선택합니다.

(2) If all processes in the ready queue are CPU bound:
- 스케줄러 관점**:** Short-term scheduler는 CPU 바운드 프로세스 중 어떤 프로세스를 실행할 것인지 선택해야 합니다. CPU 바운드 프로세스는 CPU 실행 시간을 많이 필요로 하기 때문에, 시간을 분할하여 다른 프로세스에게 넘겨주는 것이 공정하게 CPU를 분배하는 데 도움이 됩니다.
- 큐 관점**:** Ready queue에는 모두 CPU 바운드 프로세스가 있으므로, Long-term scheduler가 이러한 프로세스들을 메모리에 로드한 후, Short-term scheduler가 실행할 프로세스를 선택합니다.

Q9.

SJF(Shortest Job First) 스케줄링 알고리즘은 실질적으로 구현하기가 어렵다. 그 이 유를 설명하고 해결하기 위해 시도할 수 있는 방안에 대해 자세히 설명하시오.

- SJF(Shortest Job First) 스케줄링 알고리즘은 이론적으로 가장 짧은 실행 시간을 가진 프로세스를 먼저 실행하여 평균 대기 시간을 최소화하는 효율적인 알고리즘.
- 어려움과 한계:
  - 실행 시간 예측의 어려움**:** SJF 알고리즘은 각 프로세스의 실행 시간을 정확하게 예측해야 합니다. 하지만 프로세스의 실행 시간은 일반적으로 예측하기 어렵고, 실제로는 동적으로 변할 수 있습니다.
  - **Starvation(**기아**)** 현상**:** 실행 시간이 긴 프로세스가 계속해서 짧은 프로세스에게 밀려나 대기할 경우, 실행 시간이 긴 프로세스가 오랜 시간 동안 CPU를 할당받지 못하고 기아 현상이 발생할 수 있습니다.
- SJF 알고리즘 개선을 위한 방안:
  - 추정 실행 시간 사용**:** 실행 시간을 예측하기 어려운 경우, 추정 실행 시간을 사용하여 프로세스 스케줄링을 수행할 수 있습니다. 이를 위해 예측 모델을 사용하거나, 이전 실행의 실행 시간을 기반으로 추정할 수 있습니다.
  - 우선순위 기반 스케줄링**:** 프로세스에 우선순위를 부여하여 짧은 실행 시간과 긴 실행 시간을 고려한 스케줄링을 수행할 수 있습니다. 우선순위가 높은 프로세스가 먼저 실행되지만, 우선순위가 낮은 프로세스도 일정 시간마다 CPU를 할당받을 수 있도록 설정해야 합니다.
  - 선점형 **SJF:** 기아 현상을 방지하기 위해 선점형 SJF 알고리즘을 고려할 수 있습니다. 이 경우, 현재 실행 중인 프로세스가 실행 시간이 더 짧은 프로세스가 도착하면 중단되고, 더 짧은 실행 시간을 가진 프로세스가 실행됩니다. 이렇게 하면 기아 현상을 방지할 수 있지만, 선점에 따른 오버헤드가 발생할 수 있습니다.
  - 휴리스틱 알고리즘 사용**:** 실행 시간을 정확하게 예측하기 어려운 경우, 휴리스틱 알고리즘을 사용하여 실행 시간을 예측하고 스케줄링을 수행할 수 있습니다. 이러한 알고리즘은 실행 시간 예측을 위한 경험적인 규칙을 사용합니다.

Q10.

Time quantum (혹은 time slice)을 설명하고, 태스크의 특성과 관련하여 time quantum의 크기와 스케줄러의 성능에 관한 연관 관계를 설명하시오.

- Time Quantum은 CPU 스케줄링에서 사용되는 개념으로, 다중 프로그래밍 환경에서 각 프로세스가 CPU를 얼마나 오랫동안 사용할 수 있는지를 나타내는 시간 단위입니다. 스케줄러는 각 프로세스에게 time quantum만큼의 시간을 할당하고, 해당 시간이 지나면 다른 프로세스로 CPU를 전환.
- Time Quantum의 크기와 태스크 특성:
  - 짧은 Time Quantum: 작은 time quantum은 프로세스 전환이 빈번하게 발생하므로 오버헤드가 증가할 수 있습니다. 특히, CPU 바운드 프로세스의 경우 너무 짧은 time quantum으로 인해 CPU 할당 시간이 낭비될 수 있습니다.
  - 긴 Time Quantum: 큰 time quantum은 프로세스가 CPU를 오랫동안 점유하게 되어, I/O 바운드 프로세스가 CPU를 기다리는 시간이 길어질 수 있습니다. 또한 기아 문제가 발생할 수 있으며, 응답 시간이 길어질 수 있습니다.
- 스케줄러 성능과 Time Quantum:
  - 짧은 Time Quantum: 짧은 time quantum은 스케줄러의 응답성을 향상시킬 수 있습니다. 프로세스들이 빠르게 번갈아가며 실행되므로, 사용자가 프로세스의 빠른 응답을 느낄 수 있습니다. 그러나 오버헤드가 증가하고 CPU 할당 시간이 낭비될 수 있습니다.
  - 긴 Time Quantum: 큰 time quantum은 CPU 사용 효율성을 향상시킬 수 있습니다. 스케줄링 오버헤드가 감소하며, CPU를 점유한 프로세스가 일정 시간동안 작업을 수행할 수 있어 CPU 사용률이 높아집니다. 그러나 응답성이 감소할 수 있으며 기아 문제가 발생할 수 있습니다.

Q11.

두 프로세스들 P_1, P_2의 periods가 p_1=50, p_2=100이고, processing time은 t_1=20, t_2=35이라고 하자. 이 때, 두 프로세스들이 실시간 CPU 스케줄링(real-time CPU scheduling) 기법들 중 하나인 rate-monotonic scheduling 기법으로 스케줄링될 때, 다음의 조건에 따른 수행 과정을 Gantt chart로 보이고 스케줄링이 적절히 되는지 여부를 설명하시오.
(조건) When P_2 has a higher priority than P_1

RMS 주기가 짧은 태스크가 더 높은 우선순위를 가집니다.
- P1이 1번 실행되고, P2가 2번 실행됩니다. (P1, P2)
- 이후 P1이 다시 3번 실행됩니다. (P1)
- 그리고 P2가 4번 실행됩니다. (P2)
- 이런 식으로 반복됩니다.

```
Time:   0       50      100     150     200     250     300
Task:   P1(P2)  P1(P2)  P1      P2      P1      P2      P1
```

이러한 스케줄링은 주어진 조건에서 적절하며, P1과 P2의 주기와 실행 시간을 만족합니다. P2가 더 높은 우선순위를 가지므로 P2가 P1보다 먼저 실행되고, 주기가 짧은 P1이 먼저 실행됩니다. 이런 방식으로 두 프로세스가 주어진 주기 내에서 정상적으로 스케줄링됩니다.

Q13.

두 프로세스들 P_1, P_2의 periods가 p_1=50, p_2=80이고, processing time은 t_1=25, t_2=35이라고 하자. 이 때, 두 프로세스들이 실시간 CPU 스케줄링(real-time CPU scheduling) 기법들 중 하나인 earliest-deadline-first (EDF) scheduling 기법으로 스 케줄링될 때, 다음의 조건에 따른 수행 과정을 Gantt chart로 보이고 스케줄링이 적 절히 되는지 여부를 설명하시오.
(조건) When P_1 has a higher priority than P_2

- Earliest-Deadline-First (EDF) 스케줄링은 실시간 시스템에서 사용되는 스케줄링 알고리즘 중 하나로, 가장 빠른 마감 시간(Deadline)을 가진 태스크에 CPU를 할당합니다.
- EDF 스케줄링에서는 가장 빠른 마감 시간을 가진 태스크에 CPU를 할당하므로, P1과 P2의 마감 시간을 계산해야 합니다.
  - P1의 첫 번째 마감 시간: 현재 시간 + 주기 - 실행 시간 = 0 + 50 - 25 = 25
  - P2의 첫 번째 마감 시간: 현재 시간 + 주기 - 실행 시간 = 0 + 80 - 35 = 45

```
Time:   0       25      45      70      95      120
Task:   P1      P1      P2      P1      P2      P1
```

스케줄링이 적절하게 이루어졌습니다. P1이 P2보다 높은 우선순위를 가지므로 P1이 먼저 실행되고, P2는 P1의 마감 시간까지 대기합니다. 이후 P2가 실행되고, 다시 P1이 실행됩니다. 이런 방식으로 두 프로세스가 주어진 주기 내에서 정상적으로 스케줄링됩니다.

Q14.

Interprocess Communication(IPC) 모델은 shared memory 방식과 message passing 방식으로 나누는데, 각 방식에 대해 설명하고 특징 및 장단점을 비교하시오.

- Interprocess Communication (IPC)은 프로세스 간에 데이터를 공유하거나 통신하기 위한 메커니즘을 의미
- Shared Memory 방식: 프로세스 간에 같은 물리적인 메모리 공간을 공유하는 방식입니다. 프로세스는 공유 메모리 영역에 데이터를 쓰거나 읽을 수 있습니다.
  특징:
  - 빠른 속도: 데이터를 복사하거나 전달하지 않고 직접 메모리를 공유하기 때문에 빠른 통신이 가능합니다.
  - 복잡한 동기화: 여러 프로세스가 공유 메모리에 동시에 접근할 수 있으므로 동기화 문제를 다루어야 합니다.
  - 공유 리소스: 메모리 자원을 공유하므로 메모리 관리가 중요하며, 리소스 관리 문제가 발생할 수 있습니다.

  장점:
  - 높은 성능: 데이터 전달 오버헤드가 낮아 성능이 우수합니다.
  - 간단한 구현: 기본적인 메모리 공유 개념이 간단하며, 빠른 구현이 가능합니다.

  단점:
  - 동기화 복잡성: 공유 메모리에 대한 동기화 관리가 어려울 수 있습니다.
  - 보안 문제: 공유 메모리를 사용하면 다른 프로세스가 접근할 수 있는 보안적인 문제가 발생할 수 있습니다.
- Message Passing 방식: 프로세스 간에 메시지를 주고받는 방식으로, 메시지 큐, 파이프, 소켓 등을 통해 데이터를 전송합니다.
  특징:
  - 상대적으로 느린 속도: 데이터를 복사하고 메시지를 전송하는 데 오버헤드가 발생하므로, 공유 메모리 방식보다 느릴 수 있습니다.
  - 간단한 동기화: 메시지 전송 및 수신은 동기화에 대한 걱정이 줄어듭니다.
  - 분산 시스템: 다른 컴퓨터나 네트워크를 통해 프로세스 간 통신이 가능합니다.

  장점:
  - 보안: 메시지 패싱은 두 프로세스 간의 명시적인 통신으로 보안적인 이점이 있습니다.
  - 분산 시스템 지원: 네트워크를 통한 IPC가 가능하므로 분산 시스템에서 사용될 수 있습니다.

  단점:
  - 성능: 오버헤드가 높아 성능이 떨어질 수 있습니다.
  - 복잡성: 메시지 큐, 파이프, 소켓 등을 사용하기 위한 복잡한 코드가 필요할 수 있습니다.

Q15.

Eisenberg와 McGuire가 제안한 n개의 프로세스들에 대한 critical-section (CS) 문제의 해결방안을 세 가지 요구사항과 함께 설명하시오.

- **Mutual Exclusion (**상호 배제**):** CS에 들어가려는 프로세스는 반드시 하나씩만 들어갈 수 있어야 합니다. 즉, 어떤 프로세스가 CS에서 실행 중일 때 다른 프로세스들은 접근할 수 없어야 합니다.
- **Deadlock Freedom (**교착상태 방지**):** 시스템은 교착상태(deadlock)에 빠지지 않아야 합니다. 이것은 모든 프로세스가 어떤 시점에서 CS에 들어갈 수 있어야 하며, 어떤 프로세스도 영원히 기다리지 않아야 합니다.
- **Fairness (**공정성**):** 모든 프로세스가 CS에 공평하게 접근할 수 있어야 합니다. 한 프로세스가 다른 프로세스보다 우선권을 갖지 않아야 합니다.

Q16.

페이징 기법을 이용하는 가상 메모리 구조에서는 메모리에 해당 페이지가 없을 때, 페이지 교체를 통해 원하는 페이지를 메모리에 적재한 후 사용한다. 그러나 이런 페 이지 교체가 자주 발생하게 되면, 프로세스의 처리 시간보다 메모리의 페이지 교체 시간이 더 길어지는 쓰레싱(thrashing) 문제가 발생할 수 있다. 이와 같은 문제의 원인과 해결 방안을 설명하시오.

- 원인: 과도한 페이지 부재 (Page Fault). 쓰레싱은 주로 과도한 페이지 부재로 인해 발생. 프로세스가 메모리에 필요한 페이지를 찾을 수 없어 페이지 부재가 발생하면, 운영체제는 디스크에서 페이지를 읽어와 메모리에 적재함. 이러한 페이지 부재가 빈번하게 발생하면 페이지 교체가 반복되고 쓰레싱이 발생함.
- 해결 방안:
  - 메모리 크기 조정**:** 메모리 크기를 늘리면 페이지 부재가 줄어들고 페이지 교체가 감소하므로 쓰레싱 문제를 완화할 수 있음.
  - 워킹 셋**(Working Set)** 관리**:** 워킹 셋은 프로세스가 실행 중에 실제로 필요한 페이지의 집합. 이를 관리하고 워킹 셋 크기를 유지하는 것이 쓰레싱 방지에 도움이 됩니다. 운영체제는 프로세스의 워킹 셋을 추적하고 필요하지 않은 페이지를 제거하여 쓰레싱을 방지합니다.
  - 페이지 교체 알고리즘 개선**:** 페이지 교체 알고리즘을 최적화하여 페이지 교체 오버헤드를 줄일 수 있습니다.
  - 우선순위 설정**:** 운영체제는 프로세스의 우선순위를 설정하여 중요한 작업을 먼저 처리하도록 할 수 있습니다. 이를 통해 중요한 프로세스가 메모리를 점유하도록 우선적으로 처리할 수 있습니다.
  - 스와핑**(**화일 페이지 교체**):** 프로세스가 아예 메모리에서 제거되고 디스크로 스왑될 수 있습니다. 이렇게 하면 다른 중요한 프로세스에게 메모리를 양보할 수 있습니다. 다시 필요할 때 스왑인 되어 메모리로 복귀합니다.

Q17.

4 GB 가상메모리(virtual memory)를 갖는 시스템에서 페이지 크기(page size)가 1 MB 라고 할 때, 다음 물음에 답하시오.
(1) 페이지 테이블에 저장할 수 있는 엔트리의 수를 구하시오.
(2) 같은 프로세스를 처리하는 시스템 환경에서 페이지 크기(page size)가 4KB로 변 경된다고 할 때, I/O time과 Internal fragmentation 측면에서 예측할 수 있는 오버헤드(overhead) 또는 장점에 대해 설명하시오.

(1) 페이지 테이블에 저장할 수 있는 엔트리의 수
- 가상 메모리 크기 = 4 GB = 2^32 바이트
- 페이지 크기 = 1 MB = 2^20 바이트
- 페이지 테이블 엔트리 수 = (가상 메모리 크기) / (페이지 크기) = (2^32) / (2^20) = 2^12

따라서 페이지 테이블에는 2^12개의 엔트리가 저장될 수 있습니다.
(2) 페이지 크기가 4KB로 변경될 때 I/O time과 Internal fragmentation 측면에서의 오버헤드와 장점
- I/O Time (I/O 시간): 페이지 크기가 작아질수록 I/O 작업이 세분화되어 더 자주 발생할 것입니다. 작은 페이지 크기는 디스크에서 페이지를 로드하는 데 더 많은 I/O 작업이 필요하며, 이로 인해 I/O 대기 시간이 증가할 수 있습니다. 따라서 I/O 시간은 큰 페이지 크기에 비해 증가할 수 있습니다.
- Internal Fragmentation (내부 조각): 페이지 크기가 작을수록 내부 조각 문제가 더 빈번하게 발생할 수 있습니다. 내부 조각은 페이지의 일부가 사용되지 않고 남아있는 공간을 의미합니다. 페이지 크기가 작을수록 프로세스가 할당한 페이지 중 일부가 사용되지 않고 비어있는 상태일 가능성이 높아집니다. 이는 메모리의 낭비로 이어질 수 있습니다.

Q18.

4개의 frames을 사용하는 메모리 시스템(memory system)에서 프로세스 수행을 위한 reference string이 다음과 같을 때 (123453416), 다음 3가지 Page-replacement algorithms을 사용하여 교체되는 과정을 그림에 표현하고 발생한 페이지 부재(page faults) 수를 구하시오. (단, 교체 대상 페이지가 2개 이상인 경우에는 FIFO 기법을 적용함.)

(1) LRU replacement: 가장 오랫동안 사용되지 않은 페이지를 교체

1	**2**	**3**	**4**	**5**	**3**	**4**	**1**	**6**	**7**	**8**
1	1	1	1	5	5	5	5	6	6	6
	2	2	2	2	2	2	1	1	1	1
		3	3	3	3	3	3	3	7	7
			4	4	4	4	4	4	4	8

(2) FIFO replacement: 메모리에 가장 먼저 올라온 페이지를 교체

1	**2**	**3**	**4**	**5**	**3**	**4**	**1**	**6**	**7**	**8**
1	1	1	1	5	5	5	5	5	5	8
	2	2	2	2	2	2	1	1	1	1
		3	3	3	3	3	3	6	6	6
			4	4	4	4	4	4	7	7

(3) Optimal replacement: 가장 오랫동안 사용되지 않을 페이지를 삭제

1	**2**	**3**	**4**	**5**	**3**	**4**	**1**	**6**	**7**	**8**
1	1	1	1	1	1	1	1	6	6	6
	2	2	2	5	5	5	5	5	5	5
		3	3	3	3	3	3	3	7	7
			4	4	4	4	4	4	4	8

Q19.

Deadlock handing methods 중 하나인 Banker's algorithm을 프로세스에 할당된 자원의 수, 작업 완료시까지 필요한 최대 자원의 수, 현재 가용한 자원의 수를 가지고 설명하시오.

Banker's Algorithm은 교착 상태를 방지하고 관리하는 데 사용되는 알고리즘 중 하나

- 프로세스에 할당된 자원의 수 (Allocation): 각 프로세스가 현재 할당받은 자원의 수를 나타냅니다. 이 정보는 현재 실행 중인 프로세스가 얼마나 많은 자원을 사용하고 있는지를 나타냅니다.
- 작업 완료까지 필요한 최대 자원의 수 (Maximum): 각 프로세스가 작업을 완료하기 위해 필요한 최대 자원의 수를 나타냅니다. 이 정보는 프로세스가 완료하기 위해 얼마나 많은 자원이 필요한지를 나타냅니다.
- 현재 가용한 자원의 수 (Available): 현재 시스템에서 사용 가능한 자원의 수를 나타냅니다. 이 정보는 현재 시스템에서 얼마나 많은 자원이 사용 가능한지를 나타냅니다.

Banker's Algorithm은 교착 상태를 방지하기 위해 다음 조건을 만족하는지 확인:

- 각 프로세스의 Allocation은 Maximum을 초과하지 않아야 합니다.
- 현재 가용한 자원의 수와 각 프로세스의 최대 필요 자원의 합이 교착 상태를 방지할 수 있는 만큼 커야 합니다.

알고리즘이 이러한 조건을 만족하면 자원을 안전하게 할당하고 교착 상태를 방지합니다. 그렇지 않으면 자원 할당을 보류하여 교착 상태를 방지합니다.

Q20.

유닉스 I-node가 10개의 직접 접근 블록과 각 1개씩의 1차(single), 2차(double) 간접 접근 블록(indirect block)까지 활용한다고 할 때, 한 파일이 표현할 수 있는 최대 용량을 계산하시오. 단, 하나의 디스크 블록은 1KB 이며, 하나의 디스크 블록 주소는 4 Bytes이다. (계산기 불필요 최종 결과는 수식으로 표현 가능)

직접 접근 블록: 10개 × 1KB = 10KB
1차 간접 블록: 1개 × 1KB = 1KB
간접 블록이 가리키는 디스크 블록의 수 = 1KB / 4B = 256개
1차 간접 블록으로 접근 가능한 용량 = 256개 × 1KB = 256KB
2차 간접 블록이 가리키는 1차 간접 블록의 수 = 1KB / 4B = 256개.
2차 간접 블록으로 접근 가능한 용량 = 256KB × 256개
최대 용량 = 10KB + 256KB + 256KB × 256

Q21.

버퍼 캐시를 LRU 정책과 FIFO 정책 두 가지 방식을 사용한다고 할 때, 아래 액세스 패턴에 대해 총 액세스 타임을 계산하시오. 액세스는 블록 단위로 이루어지며, 버퍼 캐시는 총 3개의 블록을 저장할 수 있다고 가정한다. 버퍼 캐시에서의 액세스 타임은 0.1ms, 그 외의 경우는 10ms로 계산할 것.
액세스 패턴: A, B, C, D, A, E, C, B, A, D

LRU:

A	**B**	**C**	**D**	**A**	**E**	**C**	**B**	**A**	**D**
A	A	A	A	A	A	A	A	A	D
	B	B	D	D	E	E	B	B	B
		C	C	C	C	C	C	C	C
10ms	10ms	10ms	10ms	0.1ms	10ms	0.1ms	10ms	0.1ms	10ms

FIFO:

A	**B**	**C**	**D**	**A**	**E**	**C**	**B**	**A**	**D**
A	A	A	D	D	D	C	C	C	D
	B	B	B	A	A	A	B	B	B
		C	C	C	E	E	E	A	A
10ms	10ms	10ms	10ms	10ms	10ms	10ms	10ms	10ms	10ms