#study #cs

0) 목차

- CPU 스케쥴링
- 가상메모리
- 파일 시스템
- 별첨

1) CPU 스케줄링

Q) 운영체제의 스케줄링은 스레드의 CPU 사용도 조절할까요?

> 정답은 YES!
>
> 스레드는 커널 수준에서 관리하는 스레드와 유저 수준에서 관리하는 스레드로 나뉘어집니다. 여기서 커널 수준에서 관리하는 스레드는 CPU의 스케줄링 대상입니다.
>
> 그렇기 때문에 앞으로 스케줄링에서 프로세스를 언급하면 커널 수준의 스레드도 같이 포함된다고 알고 넘어가십시오.

우선순위

- PCB에 프로세스 별 우선순위를 기록하고 우선순위가 더 높은 프로세스에 자원을 더 빨리 더 많이 할당
- Window, Linux, Unix 전부 Priority 항목으로 각 프로세스의 우선순위를 확인할 수 있다
- 우선순위 선정 기준
  - CPU 활용률(utilization) : CPU 가동 시간 중 작업을 처리하는 시간 비율. 한 작업이 CPU에서 오래도록 남아서 작업을 한다면 그 작업의 우선순위가 높다
  - 입출력 집중 프로세스(I/O bound process)와 CPU 집중 프로세스 : 입출력 집중 프로세스는 실행 상태보다 입출력 작업을 완료할 때까지 기다리는 대기 상태가 더 길다. 반대로 CPU 집중 프로세스는 실행 상태가 더 길다. 그렇기에 입출력 집중 프로세스를 먼저 처리하여 후딱 대기 상태로 전환하고 그 시간에 CPU 집중 프로세스를 움직이는 게 이득이다.

스케줄링 큐

- 프로세스들이 자원 할당을 받기 위해 대기하는 줄이다. 각 줄은 원하는 자원의 종류에 따라 구별하여 배치한다.
- 큐는 대표적으로 준비 큐와 대기 큐가 있다. 준비 큐는 CPU를 이용하고 싶은 프로세스의 PCB가 서는 줄이고 대기 큐는 대기 상태에 접어든 프로세스의 PCB가 서는 줄이다.

선점형 스케줄링과 비선점형 스케줄링

- 선점형 스케줄링(preemptive scheduling) : 실행 상태에서 입출력을 위한 대기 상태 전환, 실행 상태에서 타이머 인터럽트를 맞아 대기 상태로 전환. 두 상황에 모두 수행할 수 있는 스케줄링.
  - 운영체제가 프로세스로부터 CPU 자원을 강제로 빼앗아 다른 프로세스에 할당할 수 있음
  - 한 프로세스의 CPU 독점을 막을 수 있지만 문맥 교환 횟수가 커져 오버 헤드 발생 가능성 높음
- 비선점형 스케줄링(non-preemptive scheduling) : 실행 상태에서 입출력을 위한 대기 상태 전환. 이 상황에만 수행하는 스케줄링.
  - 프로세스가 종료 또는 스스로 대기 상태에 들어가기 전까지 자원을 빼앗을 수 없는 스케줄링 방식이다.
  - 문맥 교환 횟수가 적어 오버 헤드 발생 가능성은 적지만 프로세스가 끝날 때까지 아무리 급해도 기다려야 함

CPU 스케줄링 알고리즘

- 선입 선처리 스케줄링(FCFS, First Come First Served Scheduling)
  - 큐에 삽입된 순서대로 먼저 CPU를 요청한 프로세스부터 CPU를 할당하는 스케줄링 방식
  - 먼저 삽입된 프로세스 실행 시간이 길어질 수록 뒤에서 대기하는 프로세스들의 실행이 지연됨. 이를 호위 효과(convoy effect)라 한다.
- 최단 작업 우선 스케줄링(SJF, Shortest Job First Scheduling)
  - 준비 큐에 삽입된 프로세스 중, CPU를 이용하는 시간이 가장 짧은 프로세스부터 먼저 실행
  - 비선점형 스케줄링 알고리즘으로 분류되나 선점형으로도 구현 가능
- 라운드 로빈 스케줄링(round robin scheduling)
  - 선입 선처리 스케줄링이지만, 정해진 타임 슬라이스(time slice, CPU 선점 가능 시간) 내에만 사용한다. 이것도 선점형 스켈줄링 중 하나.
  - 프로세스가 정해진 시간을 모두 사용했는 데 일처리가 안 끝났으면 문맥 교환이 발생한다
- 최소 잔여 시간 우선 스케줄링(SRT, Shortest Remaining Time Scheduling)
  - 프로세스는 정해진 타임 슬라이스 만큼 CPU를 이용하되, 남아 있는 작업시간이 가장 적은 프로세스를 CPU에서 다음에 이용할 프로세스로 낙점
- 우선순위 스케줄링(prority Scheduling)
  - 프로세스에 우선순위를 부여하고 가장 높은 우선순위를 가진 프로세스부터 실행
  - 단, 어떠한 조치가 없이 우선순위 위주로만 실행하면 우선순위가 낮은 프로세스는 무한정 대기할 수 있다. 이를 아사(starvation) 현상이라 한다.
  - 아사 현상을 방지하기 위해 오랫동안 대기한 프로세스일 수록 우선순위를 높여주는 연공서열(aging) 기법을 통해 방지 가능
- 다단계 큐 스케줄링(multilevel queue Scheduling)
  - 우선순위 별로 여러 개의 준비 큐를 사용한다
  - 우선순위가 가장 높은 큐에 있는 업무부터 먼저 전부 처리하고 그 다음 우선순위 큐에 있는 업무를 처리하는 방식
  - 우선순위가 낮은 프로세스는 여전히 길게 대기할 수 있는 단점이 있다
- 다단계 피드백 큐 스케줄링(multilevel feedback queue Scheduling)

- 다단계 큐 스케줄링과 다르게 프로세스들이 큐 사이를 이동할 수 있음
- 신규 진입 프로세스는 먼저 우선순위가 가장 높은 큐에 삽입. 타임 슬라이스 동안 실행했는 데 잔여 업무가 남은 경우에는 다음 우선순위 큐로 삽입. 이처럼 시간이 길수록 우선순위가 낮아지기에 실행 시간이 길 수록 뒤로 밀려나고 짧은 업무 시간 프로세스 위주로 먼저 돌아간다

Quiz

- 실행중인 프로세스가 할당 받은 시간을 모두 소모하여 타이머 인터럽트를 받았을 때, 어느 큐로 이동하는 가?
- 하나의 OS에서 하나의 스케줄링 알고리즘만 사용할까?

2) 가상 메모리

Q) CPU는 프로세스들이 메모리 몇 번에 저장되어 있는 지 전부 알까?

```
정답은 No!

CPU는 논리 주소를 이용하여 그 때 그 때 필요한 내용물을 기억하고 실행합니다. 메모리에 탑재된 주소를 전부 기억하고 있으면 당연하게도 레지스터 또한 메모리 만큼의 내용물이 필요하겠지요. 그럼 돈낭비...
```

물리 주소와 논리 주소

- CPU와 프로세스는 메모리 주소를 인식할 때 메모리의 실제 주소인 물리 주소가 아니라 추상화시킨 논리 주소를 기억한다
  - 논리 주소는 프로세스 내부를 0번부터 시작해서 기록한다. 메모리에 프로세스 A, B, C가 전부 다른 메모리 주소를 차지하고 있어도 논리 주소 상으로는 프로세스 A도 0번, B도 0번, C도 0번을 받을 수 있다. 그래서 논리 주소는 중복될 수 있다
- CPU와 프로세스는 논리주소를 기억하고 메모리는 물리 주소 밖에 모르는 데 서로 변환 장치가 없으면 소통할 수 없다. 이를 도와주는 게 메모리 관리 장치(MMU)이다.

스와핑과 연속 메모리 할당

- 스와핑(swapping) : 메모리에 적재된 프로세스 중 대기 상태에 있거나 장기간 사용하지 않는 프로세스들을 보조기억장치로 옮기는 방안이다.
  - 프로세스 내용물을 잠시 보조기억장치로 옮겨 메모리를 확보하고 추후에 필요가 생기면 다시 불러들이는 걸로 메모리 영역의 효율성을 증대시킨다.
- 연속 메모리 할당 : 메모리의 사용자 영역에서 프로세스와 프로세스 사이에 빈 주소 영역없이 연속적으로 저장되어 있는 걸 말한다.
  - 외부 단편화(external fragmentation) : 프로세스 실행 종료를 반복하면서 결국 프로세스 사이 사이에 메모리 빈 공간이 늘어났지만, 프로세스를 넣을 만한 공간이 아니어서 결국 빈 공간으로 계속 남는 상황을 의미한다.

가상 메모리

- 스와핑과 연속 메모리 할당은 외부 단편화 문제를 야기하며 물리적 메모리 크기에 의존하기에 결국 메모리보다 큰 프로세스를 할당할 수 없다.
- 가상 메모리(virtual memory) : 실행하고자 하는 프로그램의 일부만 메모리에 적재하여 더 큰 프로세서를 실행할 수 있게 만든 기법이다. 종류로는 페이징과 세그멘테이션이 있다.
- 세크멘테이션(segmentation) : 프로세스를 가변적 크기의 세그먼트 단위로 분할한다. 단위가 가변적이어서 프로세스 분할 및 사용 시 유연성이 있지만 외부 단편화가 발생할 수 있다.
- 페이징(paging) : 프로세스 논리 주소 공간을 페이지 단위로 나누고 물리 주소 공간을 페이지와 동일한 프레임 단위로 나눈 뒤에 페이지를 프레임에 할당한다.
    - 단위로 크기가 지정되어 있기에 연속 메모리 할당 때처럼 프로세스 크기가 들쭉날쭉 하지 않아 외부 단편화가 생기지 않는다.
    - 스와핑 기법을 적용하여 페이지 단위로 보조 기억장치로 이동할 수 있다.
    - 페이지 크기는 정해져 있기에 프로세스 크기가 그에 미치지 못하는 경우, 내부 페이지 영역이 일부 비어 있을 수 있다. 이렇게 페이지 내부에 빈 공간이 생기는 메모리 낭비 현상을 내부 단편화(internal fragmentation)라 한다.

페이지 테이블

- 페이지 테이블(page table) : 페이지와 실제로 적재된 프레임을 연결시켜주는 정보를 보관한다.
- 테이블 엔트리(PTE, page table entry) : 페이지 테이블을 구성하는 각 행을 의미한다. 행은 페이지 번호, 프레임 번호, 유효 비트, 보호 비트(r, w, x), 참조 비트, 수정 비트로 구성한다.
    - 유효 비트(valid bit) : 페이지 접근 가능 여부를 알려준다. 메모리에 페이지가 존재하면 1, 보조기억장치에 있으면 0. 보조기억장치에 있으면 바로 접근할 수 없으므로 메모리 적재가 필요하다. 만약 접근하려고 하면 페이지 폴트(page fault) 예외가 발생한다.
    - 보호 비트(protection bit) : 페이지 조작이 어디까지 가능한 지를 지정한다. r은 읽기 가능, w는 쓰기 가능, x는 실행 가능 여부를 판단한다.
    - 참조 비트(reference bit) : CPU가 페이지에 접근한 적이 있는 지 알려준다.
    - 수정 비트(modified bit) : 페이지에 데이터를 기입/수정한 적이 있는 지 알려준다. 더티 비트(dirty bit)라고도 부름. 페이지에 수정 사항이 발생했으므로 보조기억장치에 대한 쓰기 작업이 필요하다.

페이지 테이블 레지스터

- 각 프로세스의 페이지 테이블은 메모리에 쌓인다. 그렇기에 프로세스 실행 시, 해당 프로세스의 페이지 테이블이 위치한 메모리 주소를 알아야 한다. 이 위치를 기록한 레지스터가 페이지 테이블 레지스터다.
- 모든 프로세스의 페이지 테이블을 메모리에 쌓으면 메모리 접근 횟수도 많아지고 메모리 용량에도 악영향을 끼친다. 그래서 운영체제는 모든 페이지 테이블을 메모리에 쌓는 걸 지양한다.
- 메모리 접근 횟수
    - CPU의 효율성을 높이기 위해 TLB(transition look-aside buffer)라는 페이지 테이블의 캐시 메모리를 사용한다. 참조 지역성 원리에 따라 자수 사용할 법한 페이지 위주로 페이지 테이블 일부 내용을 캐시에 적재한다.

    - CPU가 접근하려는 논리 주소의 페이지 번호가 TLB에 있으면 TLB 히트, 없으면 TLB 미스라고 부르며 메모리 접근 횟수를 줄이기 위해 가급적 TLB 히트율을 올려야 한다.
- 메모리 용량
    - 계층적 페이징(hierarchical paging) : 페이지를 규칙에 따라 잘라서 필요한 페이지 테이블만 메모리에 올리고 나머지 잘린 테이블은 별도의 페이지를 이용해 기록해두고 순서대로 이용한다. 이 때, 잘린 테이블들의 위치를 기록한 테이블이 Outer 페이지 테이블이고 이를 통해 순서대로 메모리에 올리고 내리며 관리하는 것.
- 페이징 주소 체계
    - 논리 주소는 기본적으로 페이지 번호와 변위가 하나의 세트로 구성되어 있다. 페이지 번호(page number)는 말 그대로 몇 번째 페이지에 접근할 지 나타내는 것이고 변위(offset)는 접근하려는 주소가 페이지 시작 주소로부터 얼마나 떨어져 있는 지 나타낸다.
    - 논리 주소를 통해 페이지 테이블에 접근하면 페이지 테이블은 페이지 번호와 프레임 번호를 갖고 해당 페이지 번호에 맞는 프레임 번호로 안내한다.
    - 물리 주소에는 프레임 번호와 변위가 쌍으로 저장되어 있으며 프레임 번호와 변위를 통해 메모리 위치를 확인하고 해당 메모리 위치로 접근한다.

페이지 교체 알고리즘

- 메모리에 필요한 페이지만을 기입하는 방법을 요구 페이징(demand paging)이라 한다.
- 요구 페이징 진행순서
    - CPU가 특정 페이지에 접근하는 명령어 실행
    - 페이지가 현재 메모리에 있으면 CPU는 페이지가 적재된 프레임에 접근
    - 페이지가 메모리에 없으면 페이지 폴트 예외 발생
    - 페이지 폴트 발생하면 페이지 폴트 처리 루틴을 통해 해당 페이지를 메모리로 적재하고 유효 비트를 1로 설정
    - 다시 1로 돌아감
- 위와 같은 요구 페이징을 통해 지속적으로 메모리에 페이지를 적재하면 메모리가 가득찬다. 이를 방지하기 위해 일부 페이지를 보조기억장치로 보내고 어떤 페이지를 보조기억장치로 보낼 지 결정하는 알고리즘이 페이지 교체 알고리즘이다.
- 종류
    - FIFO 페이지 교체 알고리즘(First-In First-Out Replacement Algorithm) : 메모리에 가장 먼저 적재된 페이지부터 교체. 초기에 적재되어 줄곧 참조되고 있는 페이지를 교체할 우려가 있음.
    - 최적 페이지 교체 알고리즘(Optimal Page Replacement Algorithm) : 앞으로의 사용 빈도가 가장 낮은 페이지를 교체하는 알고리즘. 페이지 폴트 발생 가능성이 가장 적미나 미래의 사용빈도가 낮은 페이지를 예측하는 게 까다로움.
    - LRU 페이지 교체 알고리즘(Least Recently Used Page Replacement Algorithm) : 가장 적게 사용한 페이지를 교체하는 알고리즘. 보편적으로 사용

페이지 폴트 종류

- 메이저 페이지 폴트(mage page fault) : 보조기억장치의 입출력 작업이 필요한 페이지 폴트. CPU가 접근하려는 페이지가 물리 메모리에 없을 때 발생.

- 마이너 페이지 폴트(minor page fault) : 보조기억장치의 입출력 작업이 필요하지 않은 페이지 폴트. CPU가 요청한 페이지가 물리 메모리에는 존재하는 데 페이지 테이블 상에는 미반영.

Quiz

- 철수는 OS SW 프로그래머다. 신입이라서 아무 것도 모르고 연속 메모리 할당 형태로 OS SW를 구성했다가 상사에게 아래와 같이 까였다.
  - 아래 대화 내용을 읽고 영희의 입장에서 해 줄 수 있는 말이 무엇이며 그게 어떠한 이점이 있는지 서술하시오.

```
상사 : 철수 씨, 메모리에 뭔 완충재가 이렇게 가득해요? 조금만 프로그램 돌려도 빈 공간이 한 가득이니 아주 인터넷 쇼핑몰 차리면 고객만족도 100% 찍겠어?
철수 : 아니 그게 아니라...
상사 : 됐고 언제까지 신입 티 낼거야? 저기 하드웨어 아키텍쳐인 영희 씨한테 물어보고 고쳐봐요 좀
```

- 논리 주소가 <5, 2>, 페이지 테이블이 <5, 1>, 물리 주소가 <1, 10> 이다. 이 때, 5번 페이지는 실제로 물리 주소에 몇 번째에 위치하는 가?

3) 파일 시스템

파일과 디렉터리

- 파일(file) : 파일은 파일명, 파일 실행을 위한 정보, 파일과 관련된 부가 정보로 구성된다. 이 때, 파일과 관련된 정보를 속성(attribute) 또는 메타 데이터(metadata)라고 부른다.
  - 파일을 다루는 작업은 전부 운영체제에서 이루어지기에 응용 프로그램에서 파일과 관련된 작업을 할 때는 시스템콜을 써야 한다.
  - 프로세스는 파일 디스크립터(file descriptor)를 통해 할당 받아 사용 중인 파일들을 구분한다. 저수준에서 파일 식별을 하는 데 파일 디스크립터를 사용하며 고수준의 추상화 된 프로그래밍 언어도 결국 파일명을 통해 파일 디스크립터를 확인하고 이를 통해 파일에 대한 조작을 한다.
  - 파일 디스크립터는 입출력장치, 파이프, 소켓 또한 파일 디스크립터로 식별한다. 그렇기에 표준 입력(0), 표준 출력(1), 표준 에러(2)처럼 입출력장치를 숫자로 구분할 수 있는 것이다.
- 디렉터리(directory) : 운영체제에서 여러 파일을 관리하기 위해 추상적으로 묶어놓는 것. 윈도우 운영체제에서는 폴더(folder)라 부르며 트리 형태의 자료 구조를 채용했다.
  - 트리 구조 디렉터리에서는 최상위 디렉터리(root directory)가 가장 위에 존재하며 그 밑으로 하위 디렉터리가 표시된다. 이 때 디렉터리가 하위로 내려갈 수록 "디렉터리명/하위디렉터리명/..."으로 사선(/, slash) 구분하여 표기한다.
  - 디렉터리는 운영체제에서 디렉터리라고 별도로 정의하지 않고 파일로 구분한다. 파일에서 디렉터리에 속한 요소의 정보가 포함되어 있으면 이를 디렉터리라 부르며 이 정보는 테이블 형태로 제공한다. 이 때, 테이블의 각 행을 디렉터리 엔트리(directory entry)라고 한다.
  - 디렉터리 엔트리는 파일명과 파일이 저장된 위치를 유추할 수 있는 정보를 포함한다.

```
struct dirent {
    ino_t d_ino; // 파일의 아이노드 번호
    off_t d_off; // 현재 디렉터리 엔트리 오프셋. 엔트리 탐색 시 사용
    unsigned short d_reclen; // 현재 디렉터리 엔트리의 길이
    unsigned char d_type; // 파일 형식
    char d_name[256]; // 파일 이름
}
```

파일 할당

- 운영체제는 파일과 디렉터리를 블록(block)이라는 단위로 읽고 쓴다. 블록 하나는 보통 4096 바이트이며 블록 안에는 블록 주소도 포함된다.
- 연결 할당(linked allocation) : 각 블록에 다음 블록의 주소를 저장하여 연결리스트처럼 각각의 블록이 다음 블록을 가리키는 형태로 만드는 방식
  - 디렉터리 엔트리에는 파일명과 파일을 이루는 첫번째 블록의 주소와 블록 단위의 길이가 명시된다.
- 색인 할당(indexed allocation) : 파일을 이루는 모든 블록의 주소를 색인 블록(index block)에 모아 관리하는 방식
  - 디렉터리 엔트리에는 파일 명과 함께 색인 블록 주소가 명시된다.

파일 시스템

- 운영체제마다 각기 다른 파일 시스템을 지원하며 동일 운영체제에서도 여러 파일의 시스템을 사용할 수 있다
  - 윈도우 : NTFS, ReFS ...
  - 리눅스 : EXT, EXT2, EXT3, ... , XFS, ZFS, ...
  - 맥OS : APFS ...
- 파티셔닝
  - 보조기억장치의 영역을 구획하는 작업. 각각 나누어진 하나의 영역은 파티션(partition)이라고 한다. 파티션마다 다른 파일 시스템 사용이 가능하다.
- 포매팅(formatting) : 파일 시스템을 설정하여 어떤 방식으로 파일을 저장하고 관리할 것인지를 결정하는 것. 예를 들면 우리가 포맷을 할 때 fat32를 고르면 fat32 파일 시스템 방식을 선택한 것이다.

아이노드 기반 파일 시스템

- 해당 파일 시스템에서는 파일마다 아이노드라는 별도의 색인 블록을 가진다. 아이노드에는 각각의 번호가 부여되어 있다.
- 아이노드 기반 파일 시스템은 EXT 시리즈에서 주로 사용하는 방식이며 블록 그룹은 EXT4의 경우 다음과 같다.
  - 슈퍼 블록 : 아이노드 개수, 총 블록 개수, 블록 크기 등 전체적인 파일 시스템 정보 저장
  - 그룹 식별자 : 블록 그룹에 대한 메타데이터 저장
  - 블록 비트맵 : 현재 블록 그룹 내에서 데이터가 어떻게 할당되었는 지를 저장

    - 아이노드 비트맵 : 현재 블록 그룹 내에서 아이노드가 어떻게 할당되었는 지를 저장
    - 아이노드 테이블 : 각 파일의 아이노드 정보를 저장
    - 데이터 블록 : 각 파일의 데이터를 저장
- 하드 링크와 심볼릭 링크
    - 디렉터리 엔트리에 파일 이름과 아이노드 번호가 있으면 해당 아이노드에 접근할 수 있고 아이노드를 통해 파일 데이터에 접근할 수 있다. 동일한 파일 속성과 데이터 블록을 공유하는 형태.
    - 여기서 하드 링크(hard link)는 원본 파일과 같은 아이노드를 공유하는 파일이며 심볼릭 링크(symbolic link)는 원본 파일을 가리키는 파일이다. 윈도우에서 볼 수 있는 바로가기 형태와 유사.

마운트

- 어떤 저장장치의 파일 시스템에서 다른 저장장치의 파일 시스템으로 접근할 수 있도록 파일 시스템을 편입시키는 작업
- USB의 디렉터리를 어떻게 컴퓨터가 읽어서 그대로 표시할 수 있을까? 위와 같은 마운트 작업을 수행하기 때문이다.

Quiz

- 바둑이는 놀랍게도 코딩을 할 수 있는 천재개다. 이제는 AI 뿐만이 아니라 개한테도 일자리를 빼앗기는 말세가 도래한 건가 싶지만 그건 자차하고, 바둑이가 터미널(cmd)을 열고 오랜만에 ls -al을 입력했다. 그러자 아래와 같이 표기가 되었다.
    - 여기서 .과 ..이 무엇을 뜻하는 지 유추해보시오.

```
drwxr-xr-x  2  baduk-computer ----- --- - -- 25:12 .
drwx------+ 19 baduk-computer ----- --- - -- 25:12 ..
-rw-r--r--  1  baduk-computer ----- --- - -- 25:12 test.txt
```

- 영희는 하드웨어 아키텍쳐에서 소프트웨어로 전직하기 위해 오늘도 열심히 공부를 하고 있다. 집에서 아이노드 기반 파일 시스템 공부를 하던 도중, 깜빡 졸았는 데 키우고 있던 고양이가 신들린 듯한 자판 누르기로 원본 파일을 삭제해버렸다. 이 때, 원본 파일 데이터에 대한 어떤 링크가 살아 있어야 영희는 원본 파일 데이터에 접근할 수 있을까?

별첨

부팅

- 커널을 메모리에 적재하여 컴퓨터를 시작하는 과정이다
- CPU는 전원이 켜지면 미리 정해진 특정 주소를 읽는데 이 주소에는 바이오스(BIOS) 프로그램이 존재한다
- 바이오스는 POST(PowerOn Self Test) 과정을 통해 하드웨어 검색 후 문제사항을 확인한다. 그리고 하드웨어에 이상이 없으면 바이오스는 보조기억장치의 MBR(Master Boot Record)라는 영역으로부터 부팅에 필요한 특정 정보를 읽는다.

- 보조기억장치의 MBR에는 부트스트랩(bootstrap)이라는 프로그램이 존재하며 이를 통해 커널의 위치를 찾아 메모리에 적재한다.

가상 머신과 컨테이너

- 가상 머신(virtual machine)은 소프트웨어적으로 구현한 가상의 컴퓨터다. 하이퍼바이저(hypervisor)를 통해 기존 운영체제와 독립된 환경을 구축한다.
  - 전가상화 : 가상 머신이 호스트 시스템의 하드웨어까지 가상화한다. 그래서 호스트 시스템과 다른 OS도 사용 가능하다. 단, 가상화일 뿐, 실제로 하드웨어가 다른 건 아니기에 실제 하드웨어가 지원하지 않는 기능은 못 쓴다.
  - 반가상화 : 호스트 시스템에서 제공하는 하드웨어 기능을 그대로 사용한다. 기능 수행에 있어 안정적이지만 실제 하드웨어에 맞춰야 하기 때문에 가상화 운영체제 선택이 제한된다.
- 컨테이너(container)는 가상머신과 달리 호스트 시스템의 커널을 공유한다. 그렇기에 커널 외의 OS 기능만 가상화가 이루어지며 그 위에서 독립된 환경을 구축한다.
  - 컨테이너 오케스트레이션(container orchestration)을 통해서 다수의 컨테이너를 효율적으로 관리할 수 있으며 이에 해당하는 소프트웨어가 쿠버네티스(kubernetes)가 있다.