메모리 구조와 프로세스

안태진, 이화경, 이현제, 천현지

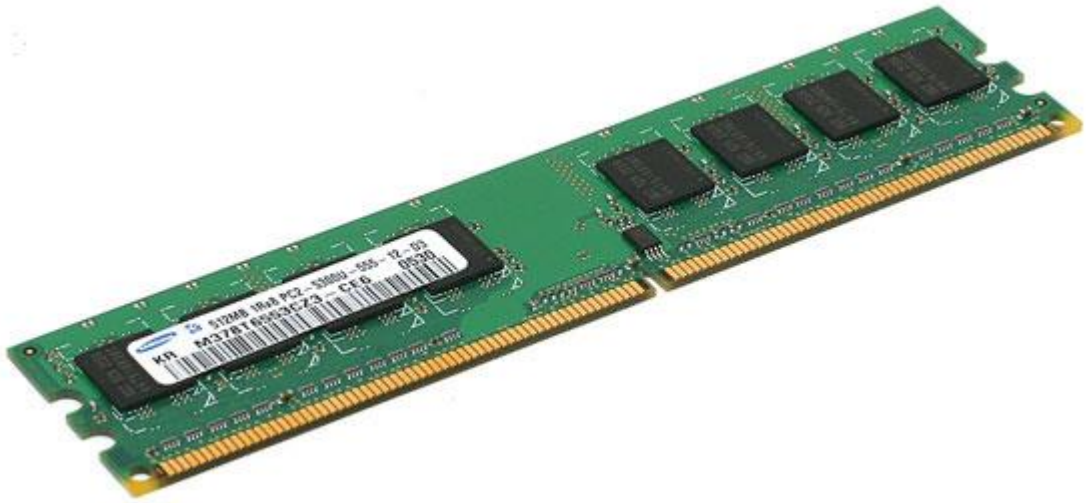
({taejin, hwakyung, hyunje, hyunji}@codecure.smuc.ac.kr)

상명대학교 보안동아리 CodeCure

목차

메모리란

- 프로그램을 실행하는 동안 정보를 저장하는 하드웨어 기억장치

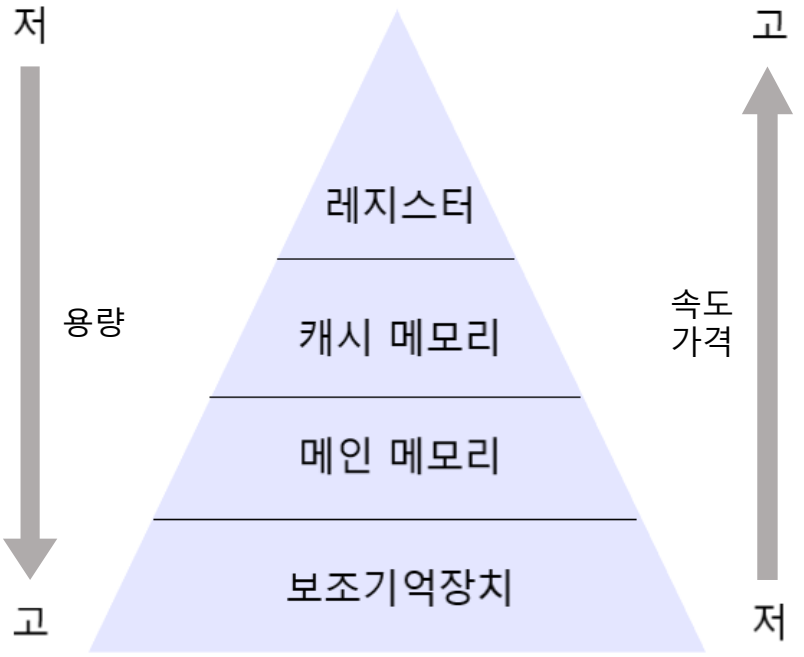


목차

메모리 종류

1. 레지스터(Register)

- 데이터를 처리하는 동안 사용할 값이나 연산의 중간 결과를 일시적으로 기억하는 CPU내에 존재하는 고속 기억장치



메모리 종류

2. 캐시 메모리(Cache memory)

- SRAM(Static Random Access Memory)
  - 전원 공급이 되는 한 저장된 데이터가 지워지지 않음
  - 재충전이 필요없음
  - 전력 소모가 많고 집적도가 낮으며 비용이 높고 속도가 빠름
- CPU와 주기억장치 사이의 속도 차이를 줄이기 위한 고속 기억장치

메모리 종류

3. 주기억장치

- 메인메모리(Main Memory)
- DRAM(Dynamic Random Access Memory)
  - 시간이 지남에 따라 축전된 전하가 감소하여 저장된 데이터가 자연히 소멸됨
  - 재충전이 필요함
  - 전력 소모가 적고 집적도가 높으며 비용이 낮고 속도가 느림
- CPU가 즉각적으로 수행할 데이터를 저장하는 휘발성 기억장치

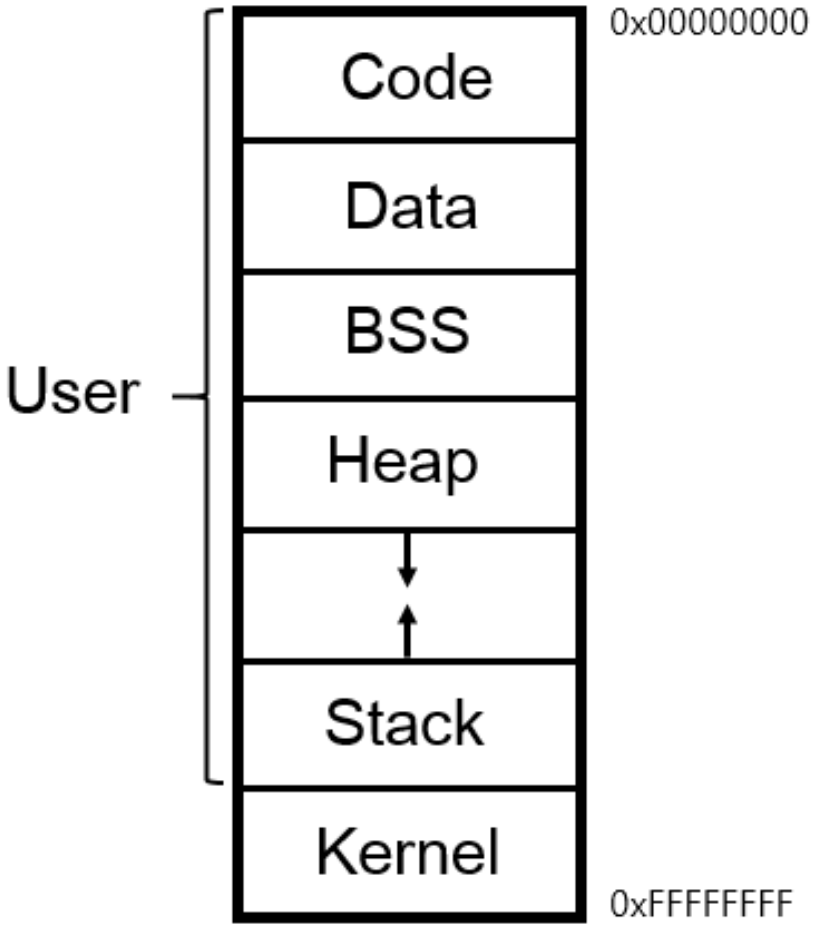
4. 보조기억장치

- SSD, 하드디스크, CD 등
- 영구적으로 데이터를 보관하는 비휘발성 기억장치

목차

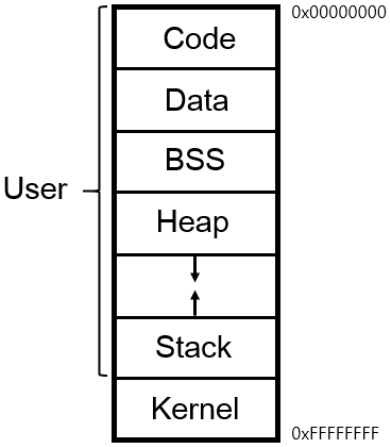
메모리 구조

- 메모리 영역


0x00000000
Code
Data
BSS
User
Heap
Stack
Kernel
0xFFFFFFFF

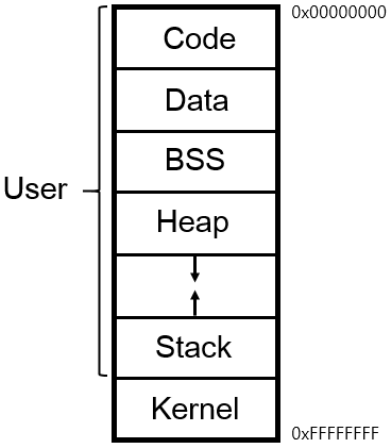
메모리 구조

- 메모리 영역
  - 커널 영역(Kernel Space)
    - 운영체제가 구동되는 영역
    - 시스템 운영에 필요한 메모리
      - 시스템 운영?
        - 메모리나 저장장치 내에서 주소공간 관리
    - 유저모드에서 접근불가능
      - 유저모드?
        - 유저 어플리케이션 코드만 실행 가능한 모드
        - 시스템 데이터, 하드웨어에 직접적인 접근이 불가능
      - 시스템 전체의 안전성과 보안을 지키기 위해 접근이 불가능함
        - 운영체제 데이터에 접근, 수정하지 못하게 함으로써 오동작을 유발하는 유저 애플리케이션이 시스템 전체의 안정성을 해치지 않게 함



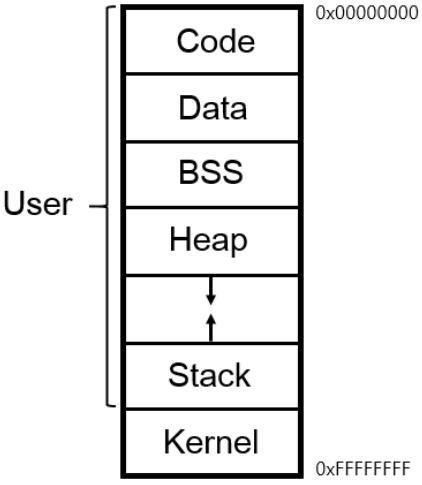
메모리 구조

- 메모리 영역
  - 유저 영역(User Space)
    - 프로그램이 동작하기 위해 사용되는 메모리
    - Code, Data, BSS, Heap, Stack영역으로 나누어짐
  - Code 영역(Code Segment)
    - 작성한 소스코드 저장
    - 읽기 전용 데이터
    - CPU가 이 영역에 있는 명령을 읽고 처리
    - 컴파일 타임에 크기 결정, 크기가 변하지 않음
      - 컴파일 타임?
        - 프로그래밍 언어를 기계어로 변경하는 과정



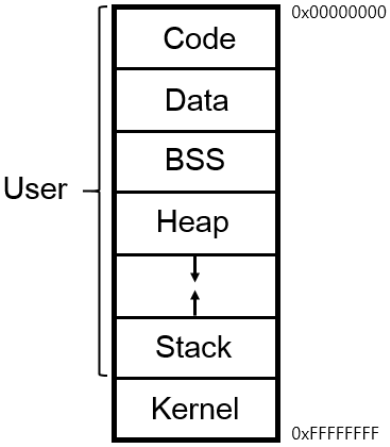
메모리 구조

- 메모리 영역
  - Data 영역(Data Segment)
    - 프로그램 시작과 동시에 할당되고 프로그램 종료 시 소멸
    - 초기화 된 전역변수, static변수
      - 전역변수?
        - 프로그램내의 어떤 지역이라도 접근할 수 있는 변수
      - static변수?
        - 생성된 지역을 벗어나더라도 소멸되지 않는 변수
    - 컴파일 타임에 크기 결정, 크기가 변하지 않음
  - BSS 영역(Block Stated Symbol Segment)
    - 초기화 되지 않은 전역변수, static변수
    - 데이터 영역과 BSS 영역을 나누는 이유
      - 실행파일(프로그램) 크기를 줄이기 위해서임



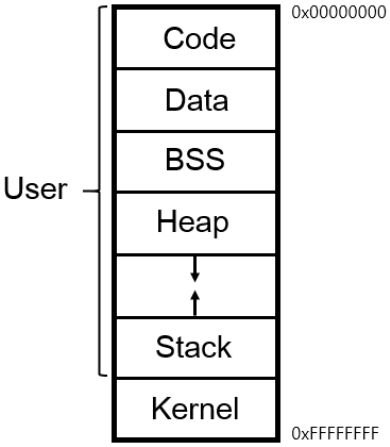
메모리 구조

- 메모리 영역
  - Heap 영역(Heap Segment)
    - 동적 할당
      - 프로그램 실행 도중 크기가 결정되는 할당 방법
    - 사용자가 직접 관리할 수 있는 메모리 영역
    - 낮은 주소에서 높은 주소로 할당
    - 런타임에 크기 결정
      - 런타임?
        - 컴파일이 끝나 생성된 실행파일(프로그램)이 실행되고 있는 때, 컴퓨터 내에서 프로그램이 기동되고 있을 때



메모리 구조

- 메모리 영역
  - Stack 영역(Stack Segment)
    - 지역변수, 매개변수, return값 등
    - 높은 주소에서 낮은 주소로 할당
    - 컴파일 타임에 크기 결정
    - LIFO(Last In First Out)
      - 후입선출, 나중에 들어온 것이 먼저 나감
    - push동작으로 데이터 저장, pop동작으로 데이터 인출
      - push동작, pop동작?
        - 스택에 데이터를 추가하거나 빼내는 명령어



메모리 구조

```c
int num1 = 10;
int num2 = 20;
int num3;
int num4;

int main(void) {
    static int num5 = 50;
    static int num6 = 60;
    int num7 = 70;
    int num8 = 80;

    int* numPtr1 = malloc(sizeof(int));
    int* numPtr2 = malloc(sizeof(int));

    printf("num1: %p\n", &num1);
    printf("num2: %p\n", &num2);
    printf("num3: %p\n", &num3);
    printf("num4: %p\n", &num4);
    printf("num5: %p\n", &num5);
    printf("num6: %p\n", &num6);
    printf("num7: %p\n", &num7);
    printf("num8: %p\n", &num8);
    printf("numPtr1: %p\n", numPtr1);
    printf("numPtr2: %p\n", numPtr2);
    printf("&numPtr1: %p\n", &numPtr1);
    printf("&numPtr2: %p\n", &numPtr2);
```

```
C:₩Users₩Jin₩Desktop>address.exe
num1: 00403008
num2: 0040300C
num3: 004063E8
num4: 004063E4
num5: 00403010
num6: 00403014
num7: 0061FEDC
num8: 0061FED8
numPtr1: 00652F20
numPtr2: 00652F30
&numPtr1: 0061FED4
&numPtr2: 0061FED0

C:₩Users₩Jin₩Desktop>
```

초기화 된 전역 변수

초기화 되지 않은 전역 변수

초기화 된 정적 변수

지역 변수

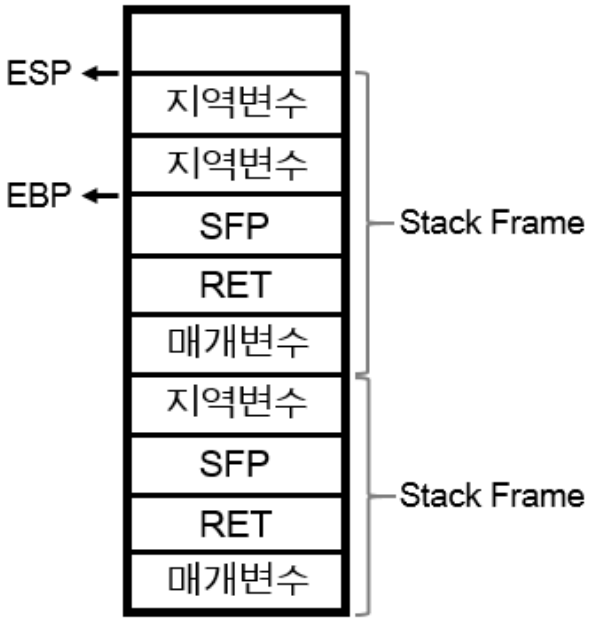
동적 할당된 주소

지역 변수

목차

메모리 구조

- 스택 프레임(Stack Frame)
  - 함수 호출 시 사용되는 스택 공간
  - ESP(Extend Stack Pointer)
    - 스택 프레임의 가장 최상부, 현재위치
  - EBP(Extend Base Pointer)
    - 스택 프레임이 시작된 주소
  - RET
    - 복귀할 함수의 주소
  - SFP(Saved Frame Pointer)
    - EBP 복귀 주소 저장



목차

프로세스란

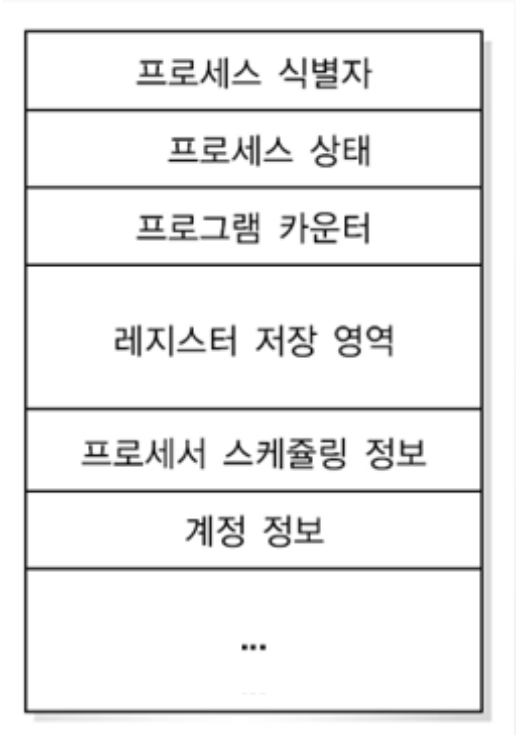
- 프로그램(Program)
  - 보조기억장치에 저장되어있는 명령어 파일
  - 생명이 없음, 실행될 수 있는 파일인 상태

- 프로세스(Process)
  - 주기억장치에 적재되어 CPU에 의해 처리되는, 실행 중인 프로그램
  - 생명이 있음, 실행되고 있는 상태

목차

프로세스 구성

- 프로세스 제어 블록(PCB, Process Control Block)
  - 프로세스에 관한 모든 정보를 가지고 있는 자료구조
  - 각 프로세스가 생성될 때마다 고유의 PCB를 가짐
  - 커널 영역에 저장됨



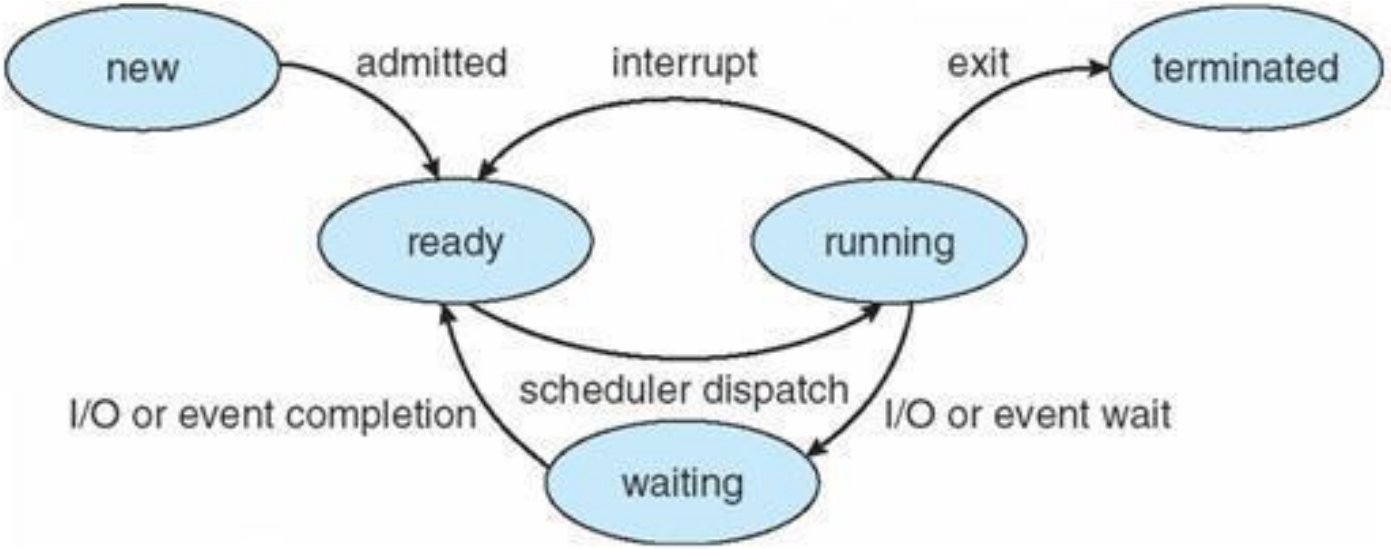
프로세스 구성

1. PID(Process Identification)
   - 운영체제가 각 프로세스를 식별하기 위해 부여된 식별번호
2. 프로세스 상태(Process State)
   - 생성(new)
     - 프로세스가 막 만들어진 상태, 메모리에 올라가기 전의 상태
   - 준비(ready)
     - 프로세스가 메모리로 올라간 상태, CPU에 올라가기 전 상태
   - 실행(running)
     - CPU를 할당 받아 실제 수행되고 있는 상태
   - 대기(waiting)
     - I/O(Input/Output) 또는 이벤트로 인해 잠시 대기상태로 전환된 상태
       - 이벤트
         - 어떤 반응이나 동작을 수행하도록 사용자가 생성시킨 동작
         - e.g., 마우스 클릭
     - 이벤트가 종료되면 준비상태로 돌아감

프로세스 구성

2. 프로세스 상태(Process State)

- 완료(terminated)
  - 실행상태인 프로세스가 종료된 상태



프로세스 구성

3. 프로그램 카운터(PC, Program Counter)
   - 다음으로 실행할 명령어가 저장된 메모리 주소 값

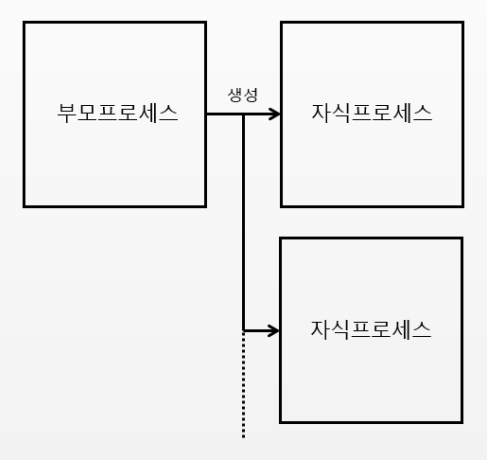
4. 스케줄링(Scheduling) 우선순위
   - 여러 개의 프로세스를 CPU에 실행될 순서를 결정한 값

5. CPU 레지스터(Register) 정보

6. 계정 정보
   - CPU 사용 시간, 계정 번호 등
     - 계정 번호
       - UID(User ID)
       - 프로세스를 실행한 특정 사용자에게 부여한 번호

프로세스 구성

7. 포인터
   - 부모 프로세스와 자식프로세스에 대한 포인터
     - 부모 프로세스
       - 프로그램 실행에 의해 처음 생성된 프로세스
     - 자식 프로세스
       - 부모 프로세스에 의해 복사되어 생성된 프로세스
   - 프로그램의 메모리 주소에 대한 포인터



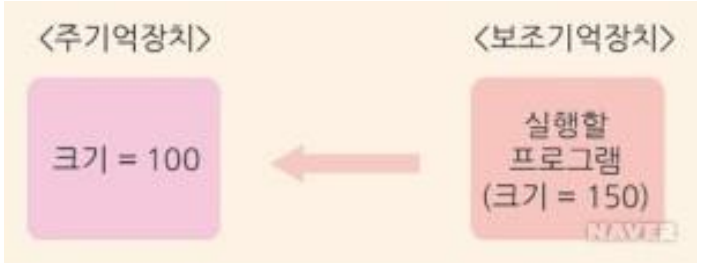
8. 입출력 정보
   - 프로세스에 할당된 입출력 장치와 파일 목록

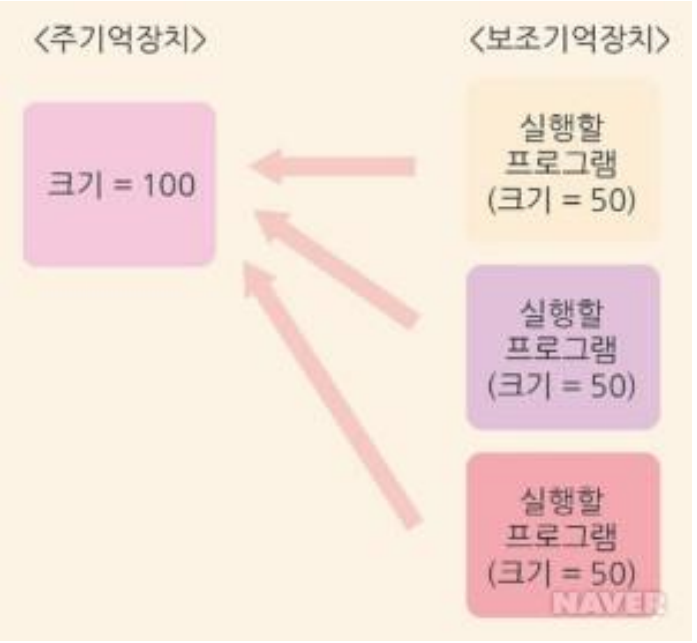
목차

- 메모리란
- 메모리 종류
- 메모리 구조
  - 메모리 영역
  - 스택 프레임
- 프로세스란
- 프로세스 구성
- 가상메모리

가상메모리

- 실행할 프로세스의 크기가 주기억장치보다 클 때 문제가 발생
- 프로세스의 필요한 부분만 주기억장치에 저장하고 나머지는 보조기억장치에 넣어 동작하는 방법으로 해결





가상메모리

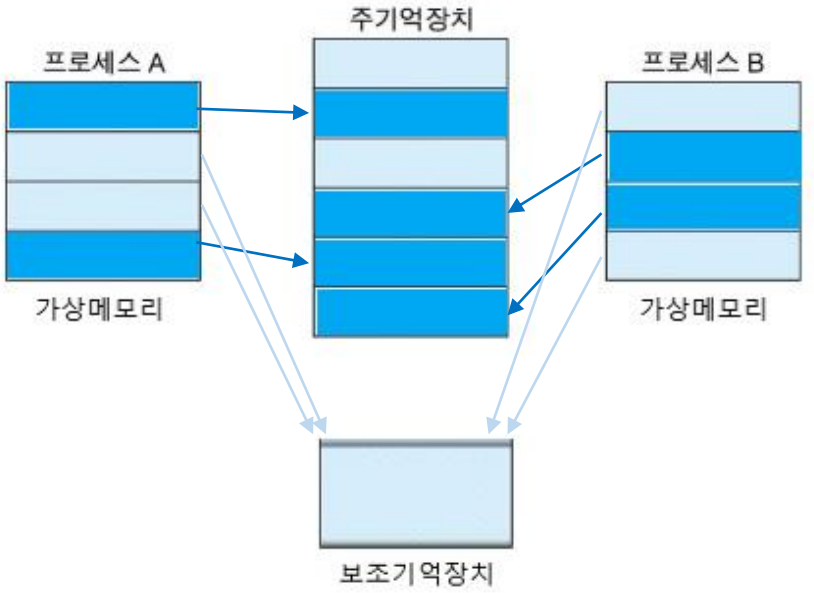
- 프로세스가 사용하는 주소가 주기억장치에 접근하는 주소와 다름

- 프로세스가 사용하는 주소

= 가상 주소 공간, 논리적 주소

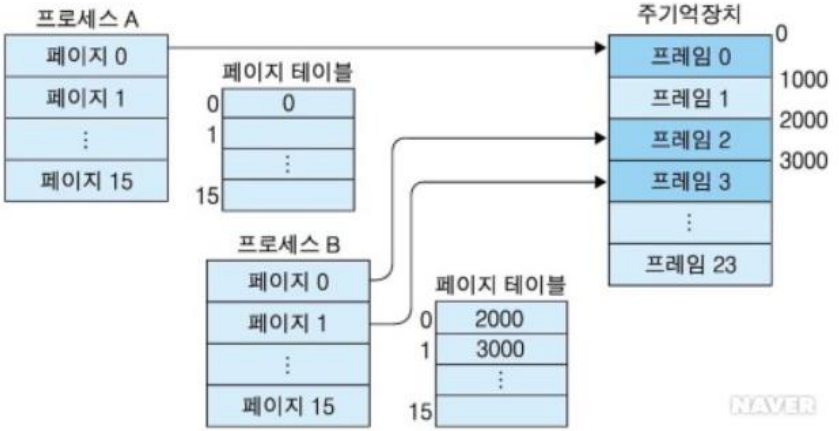
- 주기억장치에 접근하는 주소

= 실제 주소 공간, 물리적 주소



가상메모리

- 페이징(Paging)
  - 페이지 단위로 가상 주소 공간과 실제 주소 공간 사이의 데이터를 관리하는 방식
    - 페이지(page)?
      - 논리적인 의미와 관계없이 동일한 크기로 자른 블록
  - 페이지테이블(Page Table)
    - 각각의 페이지가 주기억장치의 어느 프레임에 저장될지에 대한 위치 정보를 나타냄



감사합니다!