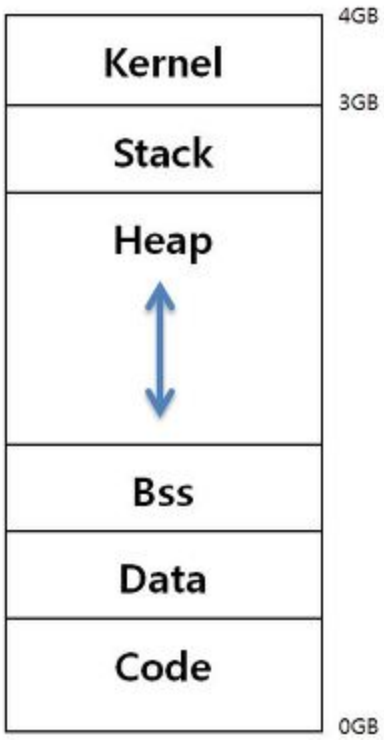
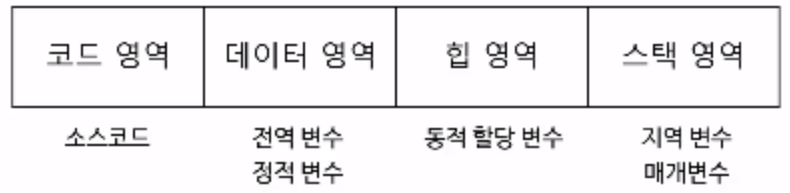
Memory

#프로세스 메모리 구조

- **메모리 구조**
  - 데이터나 프로그램을 저장하는 저장 공간은 계층 구조를 갖습니다. CPU와 가장 가까운 공간부터 레지스터 -> CPU캐시 -> 메인메모리 -> 보조기억장치 -> 외부기억장치 순이며, 이 저장 공간들은 CPU로부터 멀어질수록 데이터를 저장하는 용량이 커지고 접근하는 속도는 느려집니다.
- **가상 메모리**
  - 실제 시스템에 있는 물리적인 메모리의 크기에 상관 없이 가상 공간을 프로세스에게 제공합니다. 이런 가상 메모리는 프로세스 전체가 메모리에 적재되지 않아도 프로세스의 실행이 가능하도록 합니다.
- **메모리 구조**
  - UNIX 시스템은 실행 중인 프로세스에게 4GB의 가상 메모리 공간을 할당합니다.
  - 상위 1GB는 커널이, 하위 3GB는 사용자 프로그램이 차지합니다.

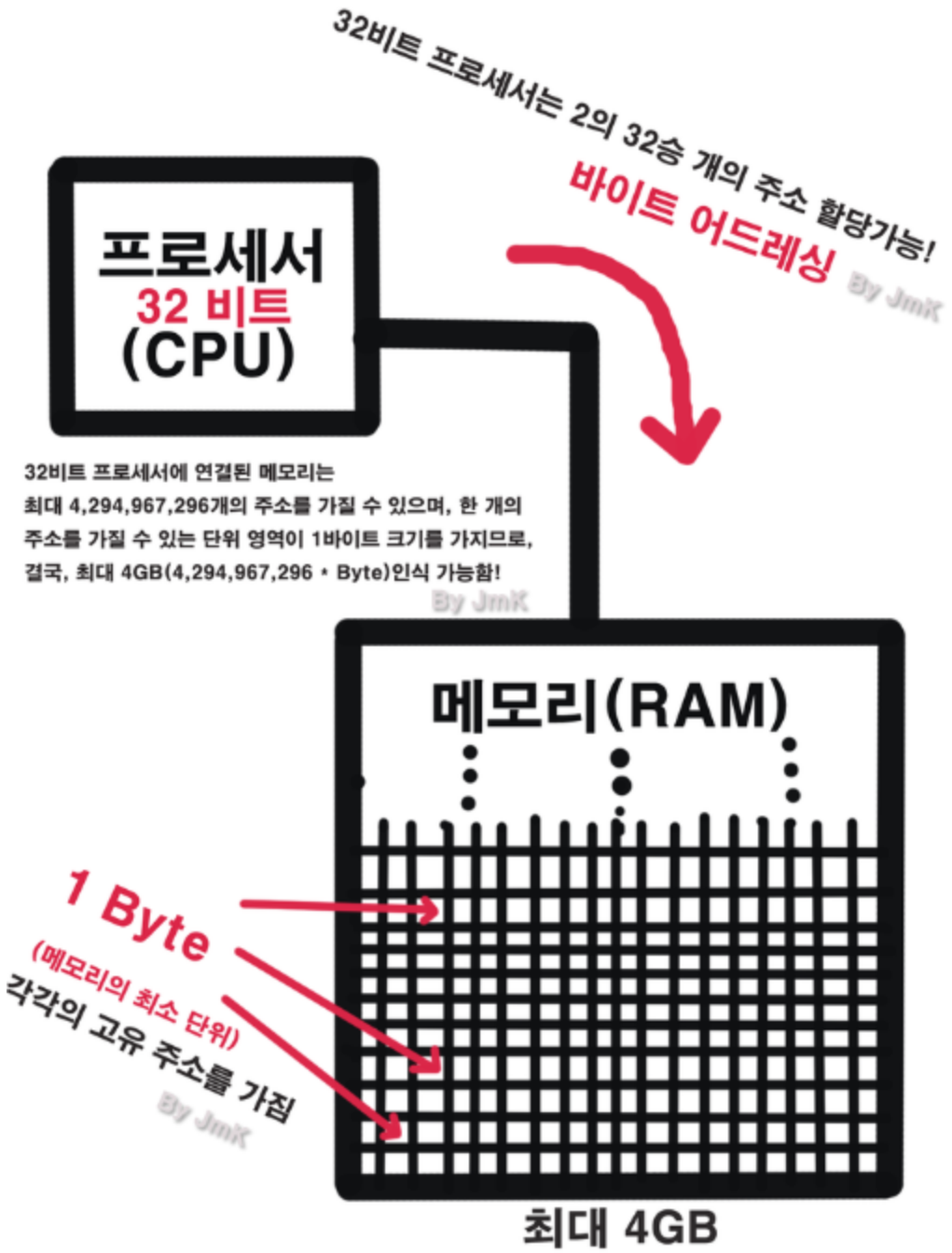


메모리 구조



1. **Stack 영역**
   a. 프로그램이 자동으로 사용하는 임시 메모리 영역으로 지역변수, 매개변수, 리턴 값 등이 잠시 사용되었다가 사라지는 데이터를 저장하는 영역입니다. 함수 호출 시 생성되고 함수가 끝나면 반환됩니다. Stack 사이즈는 각 프로세스마다 할당되지만 프로세스가 메모리에 로드될 때 Stack 사이즈가 고정되어 있어 런타임 시 Stack 사이즈를 바꿀 수 없습니다. 명령 실행 시 자동으로 증가/감소하기 때문에 보통 메모리의 마지막 번지를 지정합니다.
2. **Heap 영역**
   a. 필요에 의해 메모리를 동적 할당하고자 할 때 사용하는 메모리 영역으로 동적 데이터 영역이라고 부릅니다. 메모리 주소 값에 의해서만 참조되고 사용하는 영역입니다. 이 영역에 데이터를 저장하기 위해 C에서 malloc() 함수를 사용합니다.
3. **Data 영역**
   a. 프로그램이 실행될 때 생성되고 프로그램이 종료되면 시스템에 반환되며 전역변수, 정적변수, 배열, 구조체 등이 저장됩니다. 이 때 초기화 된 데이터는 Data 영역에 저장되고 초기화되지 않은 데이터는 BSS(Block Stated Symbol) 영역에 저장됩니다. 함수 내부에 선언된 Static 변수는 프로그램이 실행될 때 공간만 할당되고 그 함수가 실행될 때 초기화됩니다.
4. **Code 영역**
   a. 코드 자체를 구성하는 메모리 영역으로 Hex 파일이나 Bin 파일 메모리입니다. 프로그램 명령이 위치하는 곳으로 기계어로 제어되는 메모리 영역입니다.
   b. Data 영역과 BSS 영역을 구분하는 이유는 다음과 같습니다. 프로그램을 짠 뒤 컴파일하고 링크하고 이미지로 만들어 시스템의 ROM에 저장했다고 가정해볼까요? 이 때 초기화된 데이터는 초기값을 저장해야 하니 Data 영역에 저장되어 ROM에 저장됩니다. 하지만 초기화하지 않은 데이터까지 ROM에 저장한다면 큰 사이즈의 ROM이 필요한데 비용이 많이 들어 RAM에 저장하기 위해 Data 영역과 BSS 영역으로 나눈 것입니다.
5. Code, Data, BSS 영역은 컴파일 시 크기가 결정되고 Heap, Stack 영역은 런타임 시 크기가 결정됩니다.
6. Stack의 지역변수는 사용하고 소멸하므로 데이터 용량이 불확실합니다. 그렇기 때문에 밑에서부터 채워올리고 Heap은 위에서부터 채워나갑니다. 이렇게 서로 주소값을 채워나가다가 Heap에서 Stack 방향으로 영역을 침범하는 경우 HEAP overflow라고 하며 반대로 Stack에서 Heap 방향으로 영역을 침범한다면 STACK overflow라고 합니다.

출처: https://recorda.tistory.com/entry/20160503프로세스-메모리-구조 [Recorda]



https://m.blog.naver.com/jmkoo02/20199807242

컴퓨터가 인식할 수 있는 메모리의 최대 용량을 결정하는 요인: Is it the address bus size or the data bus size that ...

[Quote]

- https://blog.naver.com/jwmoon74/100099001926