Segmentation

- Base and Bounds 방식의 문제점 : Free Space가 큰 덩어리 형태로 있음(물리적 메모리 차지) -> Address Space의 크기가 맞지 않을 시 사용 불가능
- Segment : 특정 길이를 가진 Address Space의 일부분 ex) Stack, Heap, Code -> 물리적 메모리의 다른 부분에 배치되어도 문제 없음, 각 Segment 별로 Base/Bound를 가짐
- Address Translation : Physical Address = Offset + Base, Virtual Address + Base가 아님
- Segmentation Fault/Violation : 해당 공간이 아닌 주소에 접근을 시도 했을 경우, Segmentation Fault 발생(Out of bounds)

Explicit Approach

- Address Space를 Virtual Address의 상위 몇개 비트를 기반으로 여러 개의 segment로 나눔
- Stack은 위로 커짐(Code, Heap은 밑으로) -> 하드웨어의 추가적인 지원 필요 -> 하드웨어가 세그먼트가 어디 쪽으로 커지는지 확인(1:밑쪽으로, 0:위쪽으로)

Support for sharing

- 세그먼트가 Address Space사이에 공유 될 수 있음
- 추가적인 하드웨어의 지원을 통해 Code Sharing이 현재의 시스템에서 사용되고 있음 -> Protection Bit 사용(Read/Write/Execute에 관한 권한을 bit로 표시)

Fine-grained and Coarse-grained

- Coarse-grained : 작은숫자의 Segmentation ex) code, heap, stack
- Fine-grained : Address Space에 조금 더 많은 유연성 허용 -> segment table 필요

Fragmentation

- External Fragmentation : Free space 중 작은 구멍들 -> 새로 Segment 공간 할당하는게 어려움
- Compaction : 이미 있는 segment들을 재배치 하는 것 -> Free space를 통합하기 위해 -> 비용이 많이 듬(프로세스들을 모두 멈추고, 데이터를 복사해놓은 뒤, Segment Register Value도 바꿔야 함)

Free Space Management

- Splitting : 메모리에서 Free 덩어리를 찾아서 둘로 나누는 것(사용할거,Free인거) -> 요청된 할당량이 Free 덩어리 사이즈보다 작을 때
- Coalescing : 프리 덩어리 사이즈보다 큰 공간을 요청 받으면, 리스트에서 찾을 수가 없음 -> 서로 가까운 프리 덩어리끼리 합쳐서 사용한다

Header

- 메모리를 반환 할때는 메모리 공간의 사이즈를 인수로 받지 않음 -> 이 사이즈 정보는 Header Block에 저장돼 있음
- 추가적으로 할당해제를 위한 포인터, 온전성 테스트를 위한 magic 숫자도 포함 할 수 있음
- 프리 공간 = 헤더사이즈 + 사용자가 요청한 만큼의 공간 사이즈

Embedding a free list

- 메모리 할당 라이브러리는 Heap을 초기화 하고, 프리 리스트의 첫번째 엘리먼트를 free space의 처음에 놓음

- Allocation : 메모리 공간이 요청되면, 크기가 충분한 덩어리를 먼저 찾음 -> 이 덩어리를 Split 하고, 프리 덩어리의 크기를 프리 리스트에 줄임
- 대부분의 할당자들은 작은 사이즈의 힙으로 시작해서 용량이 부족해지면 OS에게 더 많은 공간을 요청함

Managing Free Space Strategies

- Best Fit : 요청 된 크기만큼 크거나 같은 덩어리를 찾음 -> 이 중 가장 작은 놈을 리턴
- Worst Fit : 가장 사이즈가 큰 덩어리를 리턴
- First Fit : 요청 된 크기만큼 크거나 같은 덩어리 중 처음 만나는 놈
- Next Fit : 크기가 맞는 덩어리를 찾되, 전에 찾기를 멈춘지점에서 시작

Segregated List

- 가장 많이 쓰는 사이즈를 기준으로, 다른 사이즈의 덩어리들을 다른 리스트에서 관리
- 얼마나 많은 메모리를 pool of memory에 줘야 특정 사이즈의 특화된 요청 처리를 할 수 있나 -> Slab Allocator
- Slab Allocator : 몇개의 객체 캐시를 할당, 이 객체들은 자주 요청 될 가능성이 높음 ex) locks
- 캐시가 부족해지면, 메인 메모리 할당자에게 더욱 요청

Binary Buddy Allocation

- 할당자가 프리 공간을 2로 계속해서 나눠서 요청받은 공간을 할당할만한 공간이 나올때까지 반복
- internal fragmentation 발생 가능성이 높음, Coalescing이 쉬워짐(작은 두블럭 합치면 그 다음 레벨의 블럭의 크기만큼 나옴)