17 Free space Management

빈 공간 관리에 대해 알아보자.

- 관리하는 공간이 고정크기 단위라면, 고정 크기 단위 리스트를 유지한다.
- malloc(), free() 등으로 가변적인 주소 공간을 다루는 경우 **외부 단편화**가 발생 (사용하는 공간 사이의 공간)

> 가변 크기를 충족하면서 빈 공간을 관리하려면?

위에 언급한 메모리 관련 함수들로 생각을 시작하자.

void *malloc(size_t size)sms 인자로 받은 size 만큼의 byte 영역을 가리키는 void pointer를 반환한다.

void free(void *ptr)은 포인터를 인자로 받아 해당 영역을 해제한다. 사이즈를 넘겨주지 않아도 알아서 사이즈를 계산해 해제한다.

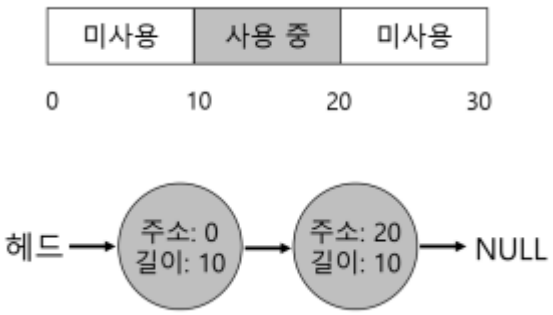
이 라이브러리들은 heap이라는 공간을 다루며, 일반적으로 링크드리스트로 관리한다.

또한, 한번 할당한 메모리는 free를 사용하기 전에는 **다른 위치로 재배치될 수 없다**고 가정한다.

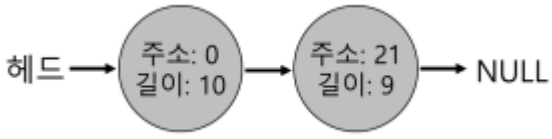
빈 공간 추적

빈 공간 리스트를 통해 위의 목적을 달성할 수 있다.

1. 주어진 상황

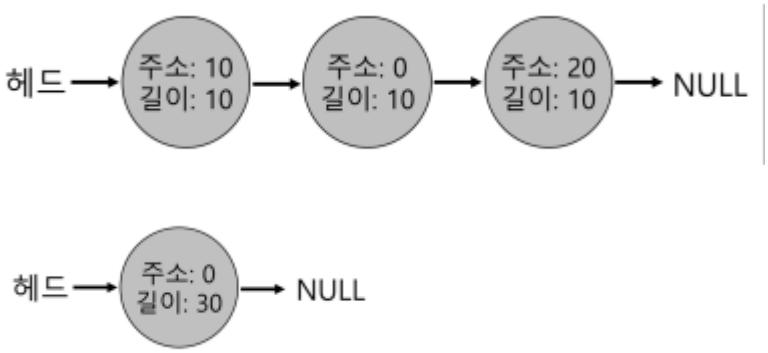


2. 분할 (빈 공간 보다 작은 메모리 요청)



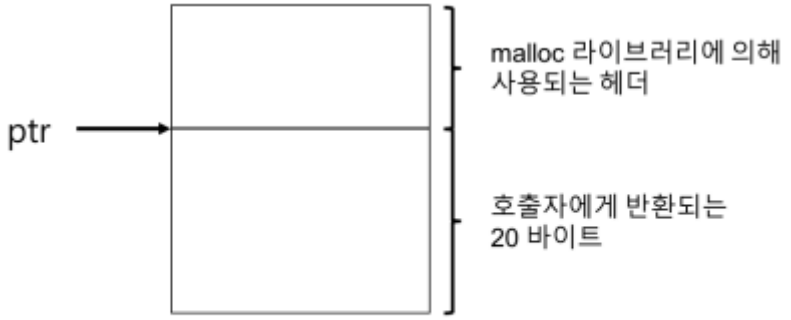
   정확하게는, 기존 20 start len 10의 공간을 20 start len1 + 21 start len 9로 분할하고 처음 청크를 반환하는 형태

3. 병합 (free 할 때 연속된 빈 공간이 존재하면 병합한다.)

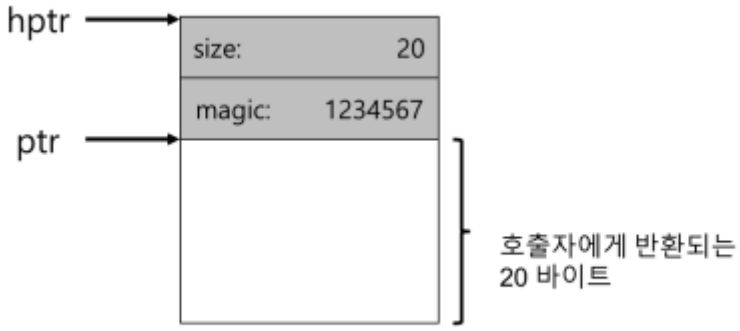


free가 size가 필요없는 이유

일반적으로 반환되는 ptr위에 hptr이 존재해서, malloc이 할당하는 size와 magic number가 포함된 말록 헤더를 운영한다. magic number는 size 기대값과 일치하는지 비교하는 안전성 검사를 수행하는 역할.



〈그림 20.1〉 **할당된 영역과 헤더**



〈그림 20.2〉 **특정 값이 저장된 헤더**

공간 할당 기본 전략

최적 적합 (Best Fit)

빈 공간 리스트를 검색하여, 요청한 크기와 같거나 더 큰 빈 메모리 청크를 찾음. 후보자 중 가장 작은 청크 반환

- 한 번만 순회화면 정확한 블럭 찾음
- 공간의 낭비 적음
- 검색 시간 많이 필요

최악 적합(Worst Fit)

가장 큰 빈 청크에서 요청된 크기 만큼 반환하고 남은 부분 유지. 커다란 빈 청크를 유지하는 것이 핵심

- 여전히 빈 공간 전체 탐색 필요
- 대부분의 경우 단편화 발생

최초 적합(First Fit)

요청보다 큰 첫 번째 블럭을 찾아 요청만큼 반환

- 속도가 빠름
- 리스트 시작에 크기 작은 객체 많아짐 -> 주소 기반 정렬을 통해 빈 공간 리스트의 순서를 관리함
- 병합이 쉽고, 단편화 감소 효과

다음 적합(Next Fit)

마지막으로 찾은 원소를 가리키는 포인터를 유지하다가, 해당 영역부터 공간 탐색해서 반환

- 리스트의 첫 부분만 사용해 단편화 일어나는 것 방지
- 균등 분산ㄴ
- 전체 탐색 x 속도 빠름

발전된 접근법

개별 리스트

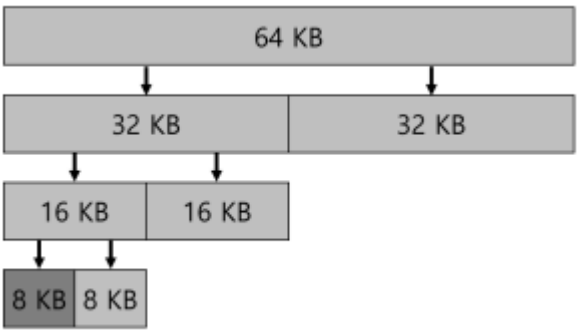
기본적인 아이디어는 특정 프로그램에 **자주 사용되는** 크기의 객체를 유지하는 별도 리스트를 유지하는 것이다.

이 외에는 일반적인 메모리 할당기가 관리한다.

이는 특정 크기의 메모리 청크를 유지함으로써 단편화 가능성을 상당히 줄이며, 복잡한 리스트 검색을 줄여주지만, 시스템에 얼마만큼의 메모리를 할당하는가 하는 문제를 지닌다. Solaris 커널에서는 슬랩 할당기(slab allocator)를 사용해서 이 문제를 해결하였다.

주로 락, 파일 시스템 아이노드 등의 커널 객체들을 빠르게 사용하기 위해 객체 캐시(object cache)를 할당해 가지고 있는다. 부팅될 때 이를 초기화함으로서 객체의 잦은 초기화와 반납의 작업을 피한다.

버디 할당



빈 공간을 2^N으로 생각하고, 가능한 한 가장 작은 공간을 찾을 때까지 2개로 쪼갠다. (내부 단편화가 자주 발생함)

해제될 때, 버디 (2개로 분할된 얘중 자기 자신이 아닌 블럭)가 비어 있으면 병합하고, 이를 위로 올라가면서 반복하면 빠르게 병합할 수 있다. 2^N 공간을 가정하기 때문에 주소공간의 비트 1개만 바꿔도 버디를 찾을 수 있다는 장점을 지닌다.