**System Programming Project 3**

**Dynamic Allocator**

담당 교수 : 박성용

이름 : 이도안

학번 : 20201610

1. **개발 목표**

Dynamic allocator를 utill과 throughput을 적절히 고려하여 효율적으로 구현한다. 수업시간이 배운 4가지의 Free list를 활용하여 구현하며, 가장 효율적인 방법을 구상하여 선택한다. 이 과제에서는 Implicit, explicit보다 성능이 높은 segmentation free list를 활용하여 free list를 삽입-삭제하는 구현을 하였다.

1. **개발 내용**

프로토타입으로 주어진 뼈대의 init, free, coalease와 extend\_heap등의 함수를 수정하여 사용하였다. Header와 footer가 모두 있는 바운드 블록을 사용하였고, 각각 헤더와 푸터를 포함한 사이즈/alloc을 저장한다. 8byte (2words) alignment를 사용하였다.

Performance 향상을 위해 extend할 때 사용되는 chunksize를 조절해가며 최적의 수를 찾았고, 추가적으로 매크로를 선언해 난해한 포인터 접근식을 쉽게 사용할 수 있도록 하였다.

추가한 매크로는 다음과 같다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

SET\_P : 위치 p 에 포인터 ptr값을 저장한다.

NEXT\_P : 바이트 단위로 포인터 ptr을 변환한다.

PREV\_P : WSIZE만큼 더해 ptr을 변환하는데, 여기서 ptr의 다음 블록에 이전 노드를 가리키는 포인터를 설정하였다.

NEXT\_NODE : seglist에서 다음 노드를 가리키는 포인터를 반환한다.

PREV\_NODE : seglist에서 이전 노드를 가리키는 포인터를 반환한다.

이 프로그램에서는 Buddy allocator 라고 불리는 2의 배수 단위의 블록 사이즈별로 나눠 할당한 segmentation list를 사용하였다.

 이와 같이 각각 포인터를 저장하는 LISTNUM(=12) 만큼의 포인터 배열을 사용하였다.

Free list를 관리하는 방법은 다음과 같다.

1. mm\_init에서 seglist의 포인터들을 각각 NULL로 할당한다.

2. insertFree 함수가 호출되면, 새로운 free block이 seglist에 추가된다.

InsertFree 함수는 넣을 block의 사이즈에 맞춰 index를 구하고, 그 index에 해당하는 리스트를 찾아 삽입하게 된다. 리스트의 노드 유무에 따라 노드를 삽입할 위치를 찾고, 포인터를 삽입한다.

3. 삽입 후 인접 블록과 포인터를 연결한다.

4. malloc을 하거나 합병을 할 때 이 Free list에서 Free block을 사용하게 되는데, 이 때 removeFree 함수를 사용해 해당 블록을 리스트에서 제거해준다. 또한 이 블록을 제거한 나머지 블록을 잘 연결해준다.

나머지 함수

void \*mm\_malloc(size\_t size)

요청된 크기의 메모리 블록을 할당한다. 요청한 크기보다 큰 가용 블록을 찾아 배치하거나, 힙을 확장한다.

static void \*extend\_heap(size\_t words)

힙을 주어진 크기만큼 확장하고, 새로 생긴 가용 블록을 초기화한다.

static void \*coalesce(void \*bp)

주변의 가용 블록들과 병합하여 하나의 큰 가용 블록으로 만든다.

void mm\_free(void \*bp)

주어진 블록을 가용 상태로 변경하고, 병합하여 가용 리스트에 추가한다.

static void \*place(void \*bp, size\_t asize)

요청된 크기의 블록을 가용 블록에 배치하고, 필요시 남은 부분을 분할하여 가용 블록으로 만든다.

1. **플로우 차트**

**이 프로그램을 간단한 플로우차트로 표현하면 다음과 같다.**

도표, 텍스트, 기술 도면, 라인이(가) 표시된 사진

자동 생성된 설명

1. **구현 결과**

작성한 코드를 mdriver -v 로 실행한 performance 및 각 테스트 데이터에 대한 correctness와 util, ops의 결과는 다음과 같다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

모든 정확도 검사에서 valid한 판정을 받았으며, 7, 8, 10을 제외한 모든 케이스에서 아주 높은 수준의 util을 보였다. Realloc 과정에서 next free와 병합하는 경우만 넣었을 때에 9, 10 케이스가 굉장히 낮게 나왔는데, perv free와도 병합하는 경우를 추가했더니 9번 케이스의 경우 99%까지 오르는 결과를 보였다.

강의자료의 베이스 코드는 implicit free list를 사용하여 55의 인덱스 퍼포먼스를 보이고, 이를 next\_fit으로 확장할 시 70점이었다. Segmentation free list와 위에 설명한 realloc의 개선을 통해 util을 53까지, throughput을 40까지 향상시킬 수 있었다.