2018년 2학기

컴퓨터 SW시스템 개론

CSED211

Lab Report #11&12

담당교수 : 김종

학번 : 20160074

학과 : 화학공학과

이름 : 고진민

POVIS ID : eric9709

--------------------------------------------------------------------------------------------------------------

<명예서약>

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

--------------------------------------------------------------------------------------------------------------

1. **구동 환경**

* 학교 프로그래밍 서버

1. **함수별 설명**

* 매크로

#define을 사용한 기본적인 매크로는 교재에 있는 것들을 그대로 채용해서 가독성과 프로그래밍을 용이하게 했다. 기본적인 상수(WSIZE, DSIZE, CHUNKSIZE)와 간단한 inline function을 매크로로 정의했다. 여기에 추가로 segregated free list의 구현 과정에서 사용하기 위한 매크로도 일부 추가했다. FWD는 앞쪽(머리, 화살표가 시작하는 부분)을 의미하며 BWD는 뒤쪽(꼬리, 화살표가 가리키는 방향)을 의미한다. 세부 설명은 주석으로 적었다.

* int mm\_init(void)

이 함수는 맨 처음 malloc 패키지를 초기화하는 작업을 한다. 이 함수는 성공적으로 초기화를 진행하면 0을, 실패한 경우는 -1을 반환한다. 먼저 16바이트(4 block)의 메모리를 mem\_sbrk 함수를 이용해 heap\_listp에 할당한다. 처음 할당하는 이 블록들은 맨 처음의 padding block(unused)와 prologue block, epilogue block이다. 이후 해당 블록에 맞는 정보(size와 allocated 여부)를 삽입하고, heap\_listp를 올바른 위치로 옮긴다. 이후 우리가 segregated free list로서 사용할 free\_list를 초기화하고 CHUNKSIZE만큼 heap을 확장한다. 이 과정을 통해 초기화를 완료할 수 있다.

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

이 함수는 size를 입력받아 메모리를 할당하고, 할당된 블록의 payload의 주소를 반환한다. 먼저 size가 0인 경우는 무시하고, size가 8 이하인 경우는 그냥 할당 크기인 asize를 16으로 고정한다. 이 16byte가 할당되는 최소한의 양으로, header와 footer가 있으면서 블록의 크기를 짝수로 해야 하기 때문이다. 8보다 큰 경우는 ALIGN 매크로를 이용해 overhead를 추가한 후 가장 가까운 8의 배수로 round up 해 asize를 정한다. 이후 이렇게 정한 asize를 ssize에 대입하고, 올바른 위치의 segregated list에서 free block을 찾는다. 할당할 수 있는 자리를 찾지 못한 경우(bp가 NULL인 경우)에는 extend\_heap를 이용해 heap을 확장한 후 place 함수를 호출한다. 이후 새로이 할당된 bp를 return한다.

* void mm\_free(void \*ptr)

이 함수는 포인터를 입력받아 해당 위치의 블록을 free block으로 만드는 함수이다. 먼저 size 변수에 해당 메모리의 크기를 받은 뒤, header와 footer의 bit를 free로 바꿔준다. 이후 해당 주소와 블록 크기를 insert\_node 함수에 전달해 free list에 free 되는 부분을 추가하고, coalesce 함수를 호출해 free block을 합친다.

* void \*mm\_realloc(void \*ptr, size\_t size)

이 함수는 최소 size만큼의 데이터가 할당된 부분의 포인터를 반환하도록 한다. 만약 입력받은 ptr가 NULL인 경우, malloc과 같고 size가 0인 경우는 free와 같다. 둘 다 아닌 경우는 입력받은 old block의 ptr의 size를 입력받은 값으로 바꾸고, 새로운 주소를 반환해야 한다. 이 주소는 원래와 같을 수도, 다를 수도 있다.

먼저 코드에서는 ptr가 NULL인 경우와 size가 0인 경우에 대해 예외처리를 진행했다. 이후 nsize에 입력받은 size에 대해 malloc에서 했던 것과 똑 같은 방법으로 block 크기를 변환해 입력한다. 이후 blockbuf에 현재 블록의 크기와 새로이 할당해야 하는 nsize를 비교해 뺀 값을 저장한다. 만약 이 값이 음수이면 이 위치에 그대로 저장을 할 수 없다는 의미가 되므로 별도의 과정을 거쳐야 한다. 만약 다음 block이 free인 상황이라면, 부족한 공간의 양을 확인해 다음 free block과 비교한다. 그래도 부족하다면, heap을 추가로 할당한다. 이후 heap에 메모리가 할당되어야 하므로 delete\_node를 호출한 후 header와 footer의 정보를 갱신한다. 만약 다음 블록이 free block이 아닌 경우는 새로운 위치에 malloc 함수를 이용해 데이터를 할당하는 수밖에 없다. 이후 memcpy 함수로 데이터를 복사하고, 기존의 ptr는 free 작업을 진행한다. 만약 blockbuf의 값이 음수가 아닌 경우는 realloc 이후에도 같은 주소일 것이므로 복잡한 과정 없이 현재 위치의 포인터를 반환하면 된다.

이 이후로는 교재에 있는 helper function을 모델로 삼아 일부 변형을 가해 사용했다.

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

이 함수는 입력받은 words의 크기만큼 heap을 추가로 할당해 확장하는 함수이다. 먼저 size에 ALIGN 함수를 이용해 새로운 크기를 할당한 후, mem\_sbrk 함수를 호출해 heap을 확장한다. mm\_init 함수에서 진행했던 것과 비슷한 과정을 거친다. Header와 footer, epilogue header의 값을 새롭게 바꿔준 수, 새로운 free node이므로 insert\_node를 호출한다. 이후 주변의 빈 공간과 합치는 과정을 거치는 coalesce 함수를 호출한 후 return한다.

* static void \*coalesce (void \*bp)

이 함수는 주변의 free된 부분을 확인해 heap을 효율적으로 관리하기 위한 함수이다. 포인터를 입력받아 주변 블록의 상황을 확인하고, 경우에 맞는 작업을 수행한다. 먼저 변수에 이전과 다음 블록의 할당 여부를 저장한다. 먼저 양 옆의 블록이 모두 할당된 상태이면 할 일 없이 기존의 포인터를 반환한다. 만약 다음 블록이 할당되지 않아 free인 경우, 먼저 현재와 다음의 블록을 free list에서 제거한 후 현재와 다음 블록의 크기를 합해 size에 저장한다. 이후 header와 footer를 갱신한다. 만약 이전 블록이 할당되지 않아 free인 경우, 역시 두 블록을 free list에서 제거한 후 size를 갱신한 뒤 header와 footer를 갱신한 후, bp를 이전 블록의 것으로 바꿔줘야 한다. 만약 앞뒤 블록이 모두 비어있다면 세 블록 보두 free list에서 제거한 후, size 갱신 후 header와 footer를 갱신하고 bp를 갱신한다. 이후 새로이 바뀐 free block을 list에 추가한 후 bp를 return한다.

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

이 함수는 입력받은 위치 bp에 asize만큼의 데이터를 할당하는 역할을 한다. 먼저 free list에서 bp에 있는 블록을 제거하고, 현재 위치의 size와(csize) 할당해야 하는 asize의 차이를 구한다. 이 때 우리는 항상 csize가 더 크다고 생각한다. 만약 이 값이 16보다 작다면, 새로운 부분을 split할 수 없으므로 현재 위치의 header와 footer를 allocated로 바꾼다. 만약 16 이상인 경우는 split이 가능하므로, split을 진행한다. 이 경우, 새롭게 생성된 free block을 list에 추가한다. 이후 할당된 블록의 주소를 return한다.

* static void insert\_node(void \*ptr, size\_t size)

이 함수는 segregated free list에 새로운 free block을 추가하는 함수이다. 이번 과제에서 사용한 segregated free list는 n번째 list에 블록 크기가 2n-1 이상 2n 미만인 메모리가 데이터 크기의 오름차순으로 들어가도록 구현했다. List는 linked list로 구현했고, 편의를 위해 양방향 linked list로 구현했다. 기본적으로 구현한, heap에 할당되는 블록의 모양은 다음과 같다. 상자 하나가 4 byte라고 생각하면 된다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Header | BWD list | FWD list |  | … |  | Footer |

앞서 언급한 매크로 역시 이에 맞게 구현되었다. 먼저 size를 이용해 올바른 list의 위치를 찾는다. 이후 이 list 내에서도 데이터가 오름차순으로 정렬되어 있으므로, 올바른 위치를 찾아 쭉 이동한다. 이동 후, 4가지 경우에 대해 다른 방법으로 node를 추가한다. List의 중간 지점, 처음, 마지막에 대해서 각각 다른 방법으로 서로를 가리키는 방향을 갱신해야 하며 마지막 경우는 list가 비어있던 상황이다. 이러한 상황의 판단은 searchp와 insertp의 NULL 여부로 확인 가능하다.

* static void delete\_node(void \*ptr)

이 함수는 insert\_node와는 반대로 ptr에 있는 node를 list에서 제거하는 작업을 한다. 이 경우 역시 해당 노드(블록)의 size에 맞는 list를 찾는다. 이후, 제거하고자 하는 node가 어디에 있는가에 따라서 작업이 달라진다. 코드 순서대로 list 중간에 있는 node, 처음에 있는 node, 마지막에 있는 node를 제거하는 것이며 마지막 경우는 list에 제거하고자 하는 node 하나만 있는 경우이다.

이상의 함수를 이용해 구현한 결과는 아래 스크린샷과 같다.

